PASCAL USERS GROUP

Pascal News

NUMBER 17

COMMUNICATIONS ABOUT THE PROGRAMMING LANGUAGE PASCAL BY PASCALERS

SEPTEMBER, 1980
POLICY: PASCAL NEWS

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

* Pascal News contains all we (the editors) know about Pascal; we use it as the vehicle to answer all inquiries because our physical energy and resources for answering individual requests are finite. As PUG grows, we unfortunately succumb to the reality of:

1. Having to insist that people who need to know "about Pascal" join PUG and read Pascal News - that is why we spend time to produce it!

2. Refusing to return phone calls or answer letters full of questions - we will pass the questions on to the readership of Pascal News. Please understand what the collective effect of individual inquiries has at the "concentrators" (our phones and mailboxes). We are trying honestly to say: "We cannot promise more that we can do."

* Pascal News is produced 3 or 4 times during a year; usually in March, June, September, and December.

* 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 18.5 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.

HERE AND THERE WITH PASCAL - presents news from people, conference announcements and reports, new books and articles (including reviews), notices of Pascal in the news, history, membership rosters, 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/algorithnm 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.
All-Purpose Coupon

Pascal User's Group, c/o Rick Shaw
P.O. Box 888524
Atlanta, Georgia 30338 USA

**NOTE**

- Membership fee and All Purpose Coupon is sent to your Regional Representative.
- See the Policy section on the reverse side for prices and alternate address if you are located in the European or Australasian Regions.
- Membership and Renewal are the same price.
- Note the discounts below, for multi-year subscription and renewal.
- The U.S. Postal Service does not forward Pascal News.

<table>
<thead>
<tr>
<th></th>
<th>USA</th>
<th>Europe</th>
<th>Aust.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 year</td>
<td>$10</td>
<td>£6.</td>
<td>A$ 8</td>
</tr>
<tr>
<td>2 years</td>
<td>$18</td>
<td>£10.</td>
<td>A$ 15</td>
</tr>
<tr>
<td>3 years</td>
<td>$25</td>
<td>£14.</td>
<td>A$ 20</td>
</tr>
</tbody>
</table>

[ ] Enter me as a new member for:
[ ] Renew my subscription for:
[ ] Send Back Issue(s)

[ ] My new address/phone is listed below
[ ] Enclosed please find a contribution, idea, article or opinion which is submitted for publication in the Pascal News.

[ ] Comments: _________________________________ 

_______________________________

_______________________________

_______________________________

ENCLOSED PLEASE FIND: A$  

CHECK no. ___________

NAME _________________________________

ADDRESS _________________________________

PHONE _________________________________

COMPUTER _________________________________

DATE _________________________________
JOINING PASCAL USER'S 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"); we will not bill you.
- Please do not send us purchase orders; we cannot endure the paper work!
- When you join PUG any time within a year: January 1 to December 31, you will receive all issues of 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.

- American Region (North and South America): Send $10.00 per year to the address on the reverse side. International telephone: 1-404-252-2600.
- European Region (Europe, North Africa, Western and Central Asia): Join through PUG (UK). Send £5.00 per year to: Pascal Users Group, c/o Computer Studies Group, Mathematics Department, The University, Southampton SO9 5NH, United Kingdom; or pay by direct transfer into our Post Giro account (28 513 4000); International telephone: 44-703-559122 x700.
- Australasian Region (Australia, East Asia - incl. Japan): PUG(AUS). Send $AIO.OO per year to: Pascal Users Group, c/o Arthur Sale, Department of Information Science, University of Tasmania, Box 252C GPO, Hobart, Tasmania 7001, Australia. International telephone: 61-02-23 0561 x435

PUG(USA) produces Pascal News and keeps all mailing addresses on a common list. Regional representatives collect memberships from their regions as a service, and they reprint and distribute Pascal News using a proof copy and mailing labels sent from PUG(USA). Persons in the Australasian and European Regions must join through their regional representatives. People in other places can join through PUG(USA).

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. Renewing for more than one year saves us time.

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.
  (A few copies of issue 8 remain at PUG(UK) available for £2 each.)
- Issues 9 .. 12 (September, 1977 - June, 1978) are available from PUG(USA) all for $15.00 and from PUG(AUS) all for $A15.00
- Issues 13 .. 16 are available from PUG(USA) all for £10; from PUG(AUS) all for $A15.00; and from PUG(USA) all for $15.00.
- Extra single copies of new issues (current academic year) are: $5.00 each - PUG(USA); £3 each - PUG(UK); and $A5.00 each - PUG(AUS).

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 18.5 cm. wide) form.
- All letters will be printed unless they contain a request to the contrary.
<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>POLICY, COUPONS, INDEX, ETC.</td>
</tr>
<tr>
<td>1</td>
<td>EDITOR'S CONTRIBUTION</td>
</tr>
<tr>
<td>3</td>
<td>HERE AND THERE WITH Pascal</td>
</tr>
<tr>
<td>7</td>
<td>Tidbits</td>
</tr>
<tr>
<td>9</td>
<td>Pascal in the News</td>
</tr>
<tr>
<td>9</td>
<td>Gossip</td>
</tr>
<tr>
<td>9</td>
<td>Books and Articles</td>
</tr>
<tr>
<td>23</td>
<td>Book review: &quot;Pascal with Style: Programming Proverbs&quot;</td>
</tr>
<tr>
<td>24</td>
<td>Review of all back issues of Pascal News (1..16)!!</td>
</tr>
<tr>
<td>27</td>
<td>Pascal Users Group finances: 1978-1979</td>
</tr>
<tr>
<td>28</td>
<td>Computer systems represented by PUG -- a summary</td>
</tr>
<tr>
<td>29</td>
<td>APPLICATIONS</td>
</tr>
<tr>
<td>29</td>
<td>Corrections to the XREF program (PN#17)</td>
</tr>
<tr>
<td>30</td>
<td>Pascal-S Subset Pascal written in Pascal</td>
</tr>
<tr>
<td>41</td>
<td>Notes on System Dependent Code in Pascal-S &amp; Pascal-I</td>
</tr>
<tr>
<td>44</td>
<td>LISP Lisp Interpreter written in Pascal</td>
</tr>
<tr>
<td>48</td>
<td>ARTICLES</td>
</tr>
<tr>
<td>49</td>
<td>&quot;An Implementation of NEW and DISPOSE Using Boundary Tags&quot; -- Branko J. Gerovac</td>
</tr>
<tr>
<td>60</td>
<td>&quot;A Simple Extension to Pascal for Quasi Parallel Processing&quot; -- Terje Noodt</td>
</tr>
<tr>
<td>67</td>
<td>OPEN FORUM FOR MEMBERS</td>
</tr>
<tr>
<td>71</td>
<td>PASCAL STANDARDS</td>
</tr>
<tr>
<td>85</td>
<td>IMPLEMENTATION NOTES</td>
</tr>
<tr>
<td>85</td>
<td>Editorial</td>
</tr>
<tr>
<td>85</td>
<td>Validation Suite Reports</td>
</tr>
<tr>
<td>112</td>
<td>Checklists</td>
</tr>
<tr>
<td>125</td>
<td>ONE PURPOSE COUPON, POLICY</td>
</tr>
</tbody>
</table>

Contributors to this issue (#19) were:

<table>
<thead>
<tr>
<th>Section</th>
<th>Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDITOR</td>
<td>Rick Shaw</td>
</tr>
<tr>
<td>Here &amp; There</td>
<td>John Eisenberg</td>
</tr>
<tr>
<td>Books &amp; Articles</td>
<td>Rich Stevens</td>
</tr>
<tr>
<td>Applications</td>
<td>Rich Cichelli, Andy Mickel</td>
</tr>
<tr>
<td>Standards</td>
<td>Jim Miner, Tony Addyman</td>
</tr>
<tr>
<td>Implementation Notes</td>
<td>Bob Dietrich, Greg Marshall</td>
</tr>
<tr>
<td>Administration</td>
<td>Moe Ford, Kathy Ford, Jennie Sinclair</td>
</tr>
</tbody>
</table>
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 R. A. Freak 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: $50.00  

Make checks payable to ANPA/RI in US dollars drawn on a US bank. Remittance must accompany application.

Source Code Delivery Medium Specification:  
9-track, 800 bpi, NRZI, Odd Parity, 600' Magnetic Tape  

( ) ANSI-Standard  

a) Select character code set:  
   ( ) ASCII  
   ( ) EBCDIC  

b) Each logical record is an 80 character card image. Select block size in logical records per block.  
   ( ) 40  
   ( ) 20  
   ( ) 10  

( ) Special DEC System Alternates:  
   ( ) RSX-IAS PIP Format  
   ( ) DOS-RSTS FLX Format

Mail request to:  
ANPA/RI  
P.O. Box 598  
Easton, Pa. 18042  
USA  
Attn: R.J. Cichelli  

Signed ____________________________

Date ____________________________

Richard J. Cichelli  
On behalf of A.H.J. Sale & R.A. Freak
Editor's Contribution

SO WHAT'S NEW

Well lots! We have extended the subscriptions of all members by 6 months. The effect of this change is that we align the subscription year to the calendar year instead of an academic year. So now, it should be easier to know when your subscription expires. Note that our policy of sending all back issues for the year has not changed. Therefore the year marked on the labels is the year through which your subscription is effective. Remember, now subscriptions expire on December 31.

Also, as you can see, if you have read the new APC, the price of Pascal News is going up. Sorry. We resisted as long as we could. But note that we offer a good price break for multiple year subscriptions. Subscribing for more than one year saves us a great deal of work. Please, please help us save paper work! The new prices will go into effect 1-January-80. Until then, we will accept renewals and subscriptions at the old price. So if you have not yet renewed, do it now, while the price is low low low! We also have a new address! (note the new APC again) You may recognize it as the return address for issues 17 and 18. The address is simple and does not include a company name. (yes the box number really does have six digits and three are 8's) I hope the new address mollifies those people who worried about vendor bias. By the way, my employer provides no support for Pascal Users Group, in any way shape or form. Which leads me to the next subject.

HELP -- I'M BEGGING

Pascal Users Group needs its own computer. It has become a necessity, to be able to maintain our ever increasing database, and do all of our record keeping. If your company can offer any type of a product for our use either as a gift, for long term use, or at a substantial discount we would like to hear from you. We are not very ambitious. Our thoughts are to secure a micro processor, a terminal, a small line printer, a hard disk, and a set of floppys. Small potatoes! Right? The system must be in place by December in order for us to be on time for the next issue. So, please, won't you call right away. (Jerry Lewis, eat your heart out) I have exhausted all my favors in Atlanta.

CHANGE OF ADDRESS -- A REAL PROBLEM

I just can not believe how many people change there address and do not inform Pascal News! The expense is phenomenal. Bulk mail is not forwardable by the post office. It costs
$.15 to send a change of address card to us, and $1.43 just in return postage if you do not. That does not include the postage to get it to you at your new address. This is a tremendous expense to PUG when 142 people "just forgot". Please help us get Pascal News to you on time. OK? So if you suspect we may have your back copies, send us a stamped self-addressed envelope with a note telling us which issues you have not recieved and we will give you your copies or a new set, no questions asked. Simple, right?

THE GOOD STUFF -- WHAT'S IN THIS ISSUE

As usual, we have a gigantic "HERE AND THERE" section this issue. It is chock full of feedback from the readers. If you put anything on the "comments" section of the APC or sent anything to me or John that was not a letter, it ends up here. So keep up the notes and comments.

I would also like to call your attention to the section on "BOOKS AND ARTICLES" if you are looking for some side reading on Pascal there are over 300 citings. Wow! And Rich has collected together a very complete list of the text books available on subject of Pascal. If your favorite is not there please drop us a line on an APC. OK?

Since Andy Mickel has a few spare moments lately, he has contributed 3 fine tidbits of information. The first is a thumbnail review of all the back issues of Pascal News (1..16). Second, he has rolled up the 78-79 finances. And third, is a summary of all the machines represented by the PUG membership, derived from the old APCs. Very interesting.

The "APPLICATIONS" section contains Wirth's Pascal-S, the subset Pascal compiler. It has been around for a while but many new users have never seen it. We also have included a LISP interpreter, for those who need the power and flexibility?! Enjoy.

The "ARTICLES" are really great too. Both show a solid approach to making a good thing better.

Jim Miner reports on the standards turmoil. The facts are laid out, and testimony from both sides is presented. You be the judge. And Let us know what you think.

And finally "IMPLEMENTATION NOTES". Forty pages of them. Note IBM's official entry. 'Nuff said.

Hope you like it.

Rick
Here and There With Pascal

TTTTTT

Peter C. Akwai, IBM Kst. 3787, Postfach 33 09,6000 Frankfurt/M West Germany: "We are willing to assume some of the unassigned Pascal Newsletter work caused by Andy Michel's retirement. Let us know what we can do to help. Pasteup, Selective composer facilities available, some graphics/cartooning, etc." (*79/05/05#)

Haim Avni, Givat Brenner, Israel 69996: "We are a rather new software group, very keen Pascalers and eager to have this line of communication with other Pascal users." (*80/05/09#)

David P. Babcock, 508 First Street, Alamosa, CO 81101: "Disappointed to note address is now DEC. Please try to maintain at least a semblance of independence in any case." (*80/01/20#)

John V. Baxter, 1830 Avenida del Mundo, Apt. 1710, Coronado, CA 92118 is using Pascal on an Apple at home, and also uses "an offspring of PASCAL -- called NCR language -- in my work at NCR Corp." (*79/12/28#)

Hank Becker, Yourdon - Software Products Group, 1133 Ave. of the Americas, New York, NY 10036: "We will be distributing a Concurrent Pascal (compiler is transportable) with P-codes to run on 8080/8085/880 and eventually other [micros]." (*80/02/23#)

Paul J. Beckmann, 1907 Bohland, St. Paul, MN 55116: "PN outstanding! Thanks to Andy and the U of M Pascal Think Tank. Good luck to you, Rick, in Georgia." (*80/02/23#)

Norman Bender, 9616 Thunderbird Drive, San Ramon, CA 94583 is interested in implementations of Pascal on TRS-80. (*80/01/05#)

E. Bhaskar, 22828 76th Ave. W, Apt. #33, Edmonds, WA 98020 is using the NBS Pascal Compiler on a PDP 11/70 to generate code which is executed on a stand-alone LSI-11 for real-time applications. (*80/01/21#)

K. Breuer, Universitaet Osnabrueck, 45 Osnabrueck, Postfach 4469 uses and teaches Pascal at University, and is very much interested in getting further issues of the newsletter. (*80/01/03#)

Frank H. Brewster, 1 North Vista Ave., Bradford, PA 16701: "If you live up to Andy's standards, you'll deserve the same huge thanks we owe to him. Good luck." (*80/02/06#)

Frank Bush, Tennessee Tech. Univ., Box 5071, Cookeville, TN 38501 has just started using UCSD B-6700 Pascal. (*80/05/06#)

R. Bush, P.O. Box F, North Bend, OR 97459: "Yeah 'Applications', Validation Suite et al. Kudos to AM for service...in nasty K. Bowes really that bad?" (*80/01/17#)

Larry H. Bush, 101 South U St., Apt. 1, Lompoc, CA 93436: "I have a system running under standard CP/M with 4K... I would like to examine the latest Pascal documentation. It seems that there are so many different versions of Pascal out. Is the standard Pascal from UCSD the best one?" (*80/01/21#)

Robert Caldwell, Scientific/Humanistic Interfaces, 2939 Governor Dr., San Diego, CA 92121: "Superb job - hang in there!" (*80/01/21#)

Dan Cantley, 3423 Carpenter Rd. Lot 10, Taylanti, MI 48197: "Just found the Pascal News - it's GREAT. Learned Pascal six months ago...our Accounting Department wanted an A/R package - our system didn't have the time or space - so I wrote the A/R package on our own micro - stuck it in Accounting Department. They love the package, and I love Pascal." (*80/01/20#)

Chip Chaplin, 3660 La Jolla Village Dr., La Jolla, CA 92037: "Should have joined long ago - have worked with UCSD Pascal project for 3 years." (*80/01/02#)

Les Cline, 1235 Wildwood Ave. #361, Sunnyvale, CA 94086: "I know not what others say, but as for me, give me Pascal, or give me Assembler!" (*80/05/06#)

Roger A. Collins, 1653 Olmada St., Encinitas, CA 92024: "I have found Pascal News very informative and helpful. Brought up an interpreter (* on a Perkin-Elmer 8/32 *) but found it unworkable in our environment, am now looking for a compiler." (*80/01/23#)

Stan Crouch, Technicon Medical Information Systems Corp., 3255-1 Scott Blvd., Santa Clara, CA 95051: "I am doing a study on the feasibility of converting some on-line programs to Pascal. I need to know whether or not Pascal programs can be made re-entrant and what is required in the operating system. Also, if you have any information on ADA capabilities I would appreciate any input in that area." (*80/04/08#)

Jeff Davis, 1515-J Tivoli Court, Raleigh, NC 27604 belongs to a local Apple users group that has started a Pascal Special Interest Group with good response. (*80/02/06#)

Tony DiCenzo, Digital Equipment, MR-11/M40, Marlboro, MA 01752: "Good luck Rick - I'm sure this publication will flourish in your capable hands." (*80/02/03#)

George B. Diamond, Diamond Aerosol Corporation, R.D. #1, Glen Gardner, NJ 08826: "If we had this kind of effort in other fields we would not be a 3rd rate power." (*80/01/23#)

John Dickinson, Dept. of Elec. Engr., Univ. of Idaho, Moscow, Idaho 83843 is
running Pascal on an IBM 370/145 and an HP1000 model 40. (*80/04/01*)

M. F. Doore, 1015 E. 10th St., Long Beach, CA 90813 is a Pascal Watcher in Electrical Engineering hoping to be the owner of a Western Digital P Machine soon. (*80/03/31*)

Donald L. Dunstan, Cogitronics Corporation, 5470 N.W. Innsbrook Place, Portland, OR 97229: "Cogitronics develops software for microprocessor development systems. Currently we are working with a Genahed/Futuredata RS/85 development system and have generated a Pascal compiler for this system." (*80/01/23*)

Hank Feener, 644B Washington Ave., Ft. Lawton, Seattle, WA 98199 owns an Apple II with Pascal and would greatly appreciate "any additional information on the implementation of Pascal on the Apple II!" (*80/01/23*)

William A. Freberg, Computer Sciences Corporation, 2753 Highland Dr., Las Vegas, NV 89109: "Implementing Pascal 6000 from Zurich on CDC 6400 owned by Department of Energy at Las Vegas NV (NOS/R8 operating system)." (*80/05/06*)

Edward R. Friedman, CIMS/New York Univ., 251 Mercer Street, New York, NY 10012: "Pascal is currently being used in courses devoted to programming languages. PROSS is also popular among researchers." Versions in use are Pascal 6000 Release 3 and Pascal from Sweden. (*80/01/23*)

Stuart H. Gage, Department of Entomology, Michigan State Univ., East Lansing, MI 48824 is "currently running UCSD Pascal on a Terak 8510/a and a CIMS MP-211, along with CDC Pascal on a Cyber 750/175. Our applications deal with delivery of agricultural information using microcomputer networks with an emphasis on graphics." (*80/01/23*)

Stephen Gerke, 1646 Parkcrest Cir., #301, Reston, VA 22090 says we should consider publishing smaller but more regular PNs. Validation reports are very helpful." (*80/05/05*)

Pete Gifford, Allegheny College, Meadville, PA 16335 is running Pascal on an IBM 4331. (*79/12/26*)

Paul J. Gillian, P.O. Box 2202 C.S., Pullman, WA 99163: "Finally got my computer (a Western Digital Pascal micro-Engine) and it's great!" (*80/01/23*)

Thomas Gliventer, 127 Linden Ave., Ithaca, NY 14850: "You might be interested to know that the latest version of Ithaca Intersystems'...Pascal/2 now runs under CP/M (instead of K2) and supports real numbers and pointer variables... See Byte, Jan. '80, page 14." (*80/01/23*)

R. Steven Glaville, Silicon Valley Software, Inc., 1531 Sandpiper Ct., Sunnyvale, CA 94087 is currently implementing an MC68000 Pascal compiler (*80/02/04*)

Steven K. Harr, Ohio State University, University Hospitals, 410 W. 10th Avenue, Columbus, OH 43210: "We are currently in the process of evaluating PASCAL compilers for use at our installation. We are running VS2 Release 1.7J on an IBM 370 Model 158J with 1.5 Mbytes of memory... Any literature you may have concerning PASCAL compilers for IBM 370 computers would be extremely helpful to us at this point." (*80/01/16*)

Michael E. Harris, 407 W. Calhoun #17, Springfield, IL 62702: "I heartily agree with the PUG direction. I hope to be installing PASCAL on my Z-80 S100 system later this year. The main thing that I would like to see happen relative to PASCAL would be the establishment of an IBM/AMDHAL 370/3033/470 vendor supported standardised version of the language. Anybody out there have a Sperry-Univac/Varian V77-600 PASCAL that an individual could afford?"

Sassan Pasamgh, P.O. Box 4526, Stanford, CA 94305: "How about setting up a Pascal Program Library (a la SHARE)?" (*80/04/01*)

Thomas Hickey, 295 Garden Rd., Columbus, OH 43214: "Enjoy Pascal News very much. Have brought up Krisch-Hansen's Sequential on (Kerox) Sigma-9 limited implementation & very slow!" (*80/04/01*)

Jean Philippe Hilaire, 77 rue Vergniaud, 75013 Paris, France would like to know who supplies PASCAL compilers for Interdata 8/32, Interdata 8/16, Perkin Elmer DS 3220 and 3240. (*80/01/23*)

William T. Hole, M.D., 260 Collingwood, San Francisco, CA 94114 has Pascal/M and is hoping to "unleash the power of Pascal on my massive behavioral research observation files, which deal with premature babies in an intensive care nursery." (*80/04/23*)

Kenneth R. Jacobs, 10112 Ashwood Dr., Kensington, Maryland 20795 is using Pascal on a DEC-10 and Xitan (Z-80) (*79/02/13*)

Steve Jay, Computer Center, University of Arizona, Tucson, AZ 85721: "I am manager of software for the University's Computer Center. We provide PASCAL for use by any of our customers (*on a CDC Cyber 175 and a DEC-10*). So far, they seem happy with it." (*80/01/21*)

R. L. Jenkins, Hartman Technics, #612-815-1st St. S.W., Calgary, Alberta, Canada T2P 1J3: "We are particularly interested in PASCAL for microprocessors. As an electronics design consultancy we produce a lot of microprocessor machine code, and would prefer to leave this uninspiring task to a compiler." (*80/02/14*)

Mort Jonas, P.O. Box 390874, Miami Beach, FL 33139: "I've been using Pascal on the Apple II, and would be most interested in seeing how it would do on the validation suite, though I'm afraid I don't have time to do it myself." (*80/01/23*)

Bursa Figg, 2206 NE 19th Place #D, Seattle, WA 98115: "I am a programmer for Boeing writing my first PASCAL program to update a Boeing cost accounting data entry system." (*80/01/20*)

Les Kitchen, Computer Science Center, University of Maryland, College Park, MD 20742: "We're using National Bureau of Standards compiler (PDP-11/Unix), Naval Undersea Lab compiler and University of Wisconsin compiler (both
work!!" (*80/05/05*)
as
applications; use here is growing but not widespread; am looking forward to
OMSI Pascal V1.2
James
Robert
David
Watkins-Johnson
NOS/BE;
George A. Martinez, 654 1/2
are firmly based in
attempt to push state-of-the-art in print size reduction. My eyes are out
to
Chestnut Hill, MA
good work. You guys are just

(*80/01/04*)

(*80/02/14*)

(*80/01/21*)

(*80/05/05*)

(*80/01/21*)

(*80/01/05*)

(*80/04/06*)

(*80/01/20*)

(*80/04/03*)

(*80/01/23*)

(*80/01/23*)

(*80/04/03*)

(*80/05/06*)

(*80/05/06*)

(*80/05/05*)

(*80/02/24*)

(*80/05/05/)
Bill Norton, M.H.S. Div., Harnischfeger Corp., 4400 W. National, Milwaukee, WI 53201: "Keeping the present PUG structure and mission is the best way to go. Best of luck to Rick Shaw and friends. Can't use Pascal much right now, but want to stay current." (*80/01/21*)

Thomas J. Oliver, Blue Hills, Dewey, AZ 86327 has a micro and plans to mainly work on alpha numeric, gray scale, pictorial maps and some LANDSAT satellite algorithms." (*80/03/20*)

Ross R. W. Parlette, Chemical Systems, United Technologies, P.O. Box 358, Sunnyvale, CA 94086: "I went to a 1 day seminar to introduce Pascal; it was very helpful. We hope to have the Validation Suite ready on the VAX for DEC Pascal in Feb. '80. (*80/01/23*)

Jeff Pepper, 5512 Margaretta St., Pittsburgh, PA 15206: "Glad you exist!" (*80/02/24*)

James G. Peterson, 1446 6th St., Manhattan Beach, CA 90266: "Keep up the good work! Some form of advertising might be worthwhile, so that more people would know about PUG. I am writing a large CAD system with PASCAL at TRW DSAC." (*80/01/21*)

Gregory N. Pipert, 1200 Columbia Ave., Riverside, CA 92507: "I am using Electro Science Ind. Pascal to drive an ESL Laser system which is used to trim thick-film potentiometers." (*80/02/14*)

Fred Pospenschil, 3108 Jackson St., Bellevue, WA 98005: "I am looking for Pascal implementations on Heath BB computers" (*That's a PDP-8 architecture*) (*80/04/03*)

Hardy J. Pottinger, EE Dept., Univ. of Missouri, Rolla, MO 65401: "Keep up the good work! I am using Pascal as a microcomputer system development language." (*80/01/23*)

Fred W. Powell, P.O. Box 2543, Staunton, VA 24401: "I am now using Pascal on a TI 990/10. Thanks for such a tremendous job with Pascal News." (*80/01/07*)

James A. Powton, 113 Chaplin Cr, Toronto, Canada M5P 1A6: "I anxiously await each issue; keep up the excellent work!" (*80/02/14*)

Robert N. Pritchett, Trans-National Leasing, Inc., Box 7245, Dallas, TX 75209 is looking for Pascal for the IBM Series/1 running the EDX operating system, or for source code for a Pascal compiler/interpreter on IBM standard 8-inch single-density diskettes, 128 bytes per sector, single or double sided.

Paul Rahin, Philadelphia Health Mgmt. Corp., 530 Walnut St., 14th Floor, Philadelphia, PA 19106: "I am interested in using Concurrent Pascal to implement a real-time dispatch system for the Philadelphia fire dept. I am looking for D.G. implementations or help converting another to D.G." (*80/04/03*)

Armando R. Rodriguez, c/o S.P. Vovda, Armanspergrasse 15, 8000 Muenchen 90, West Germany: "Coming soon: I'll have all PUG software tools in diskette (8 inch, single density, one-sided) to distribute and/or exchange for other tools." (*80/01/07*)

Bennie Roeman, 864 Watertown St., W. Newton, MA 02165: "We use Pascal heavily at Framingham State College and all in-house software at Paramic, Inc...is written in Pascal. Keep up the good work!" (*80/01/21*)

Ira L. Ruben, 2104 Lincoln Dr. East, Ambler, PA 19002: "Have used Pascal to code a Floyd-Evans production metacompiler, also currently designing and coding a communications system (Univac 'DCA') in Pascal. The language is the best I have ever used for implementation except for its lack of data alignment control and packing control, which is needed when processing bit-oriented protocols. PUG is good, but it would be nice if the news came out at more predictable intervals!" (*80/01/21*)

William John Schaller, 4309 28th Ave. S., Minneapolis MN 55406: "I work for Sperry Univac. We are developing a graphics system on a color terminal (Chromatics). We are using UCSD Pascal on a 280 to accomplish this." (*80/05/05*)

G. A. Schram, Dr. Neher-Laboratories, P.O. box 421, 2260 AK Leidschendam, The Netherlands would like to know about the availability of a DEC-10 or PDP-11 Pascal cross-compiler for the M6800 or Z-80. (*79/11/07*)

Herbert Schulz, 5820 Oakwood Dr., Lisle, IL 60532: "I've been very excited about Pascal ever since reading about it in BTTS. Have had UCSD Apple Pascal since it came out and just got UCSD Pascal for our H-11/A at the Community College where I teach. Still be teaching Pascal to the faculty next term. I'd appreciate any help for that task!" (*80/04/01*)

Ted Shapito, 5110 E. Elainore Ave, Orange, CA 92669 sends word that Dr. Donald Knuth and Dr. Luis Trabt Pardo at Stanford University are working on a typesetting system, to be implemented in Pascal.

Richard Stemborski, Communications & Computer Sciences Dept., Exxon Corp., Box 153, Florham Park, NJ 07932: "I would like a copy of the listing of ALL known PASCAL implementations for micros, mini's, and mainframes." (*80/02/03*)

Seymour Stinger, Bldg. 606/M.S. K10, Hughes Aircraft Co, P.O. Box 3310, Fullerton, CA 92634: "We are offering a 12-week class on PASCAL programming to high school students using Grogoro's text. We have installed both the SLAC and HITAC compilers on our two Amdahl 470 V/8 computers. The response to this class has been overwhelming! Many students have bought the UCSD system on the Apple microcomputer." (*80/01/10*)

K R Smith, 1632 Hialeah St., Orlando, FL 32808: "Have just ordered HP/1000 (RET IVB) Pascal. I'll let you all know as I start using it." (*80/05/05*)

Jon L. Spear, 1007 S.E. 13th Ave., Minneapolis, MN 55414: "I am working with Prof. B. Sruell and G. M. Schneider on a text: Advanced Programming and Problem Solving with Pascal" which may be available from Wiley by the fall." (*80/05/06*)
With more than 56K. (*79/12/28*)

Howard White, Jr., 799 Clayton St., San Francisco, CA 94117 would like information on Pascal 8000 as developed by the Australian Atomic Energy Commission; he is especially interested in references, bibliographies, and user feedback. (*80/03/18*)

Jerome P. Wood, 6105 Harris, Raytown, MO 64133 is interested in Pascal compilers for an IBM S/370 at work. (*80/02/03*)

Stephen Woodbridge, 642 Stearns Ave., Palm Bay, FL 32905: "Please keep up the great work. #13 is my last issue and I can't get enough of it." (*79/12/28*)

R. P. Wolff, Ajax Corp., W154 E9105 Elm La., Menomonee Falls, WI, 53051: "Are any compilers available for a 'Microdata Reality or Royale' system?" (*80/01/23*)

George O. Wright, 700 7th St. SW 635, Washington, DC 20024: "Please be friendly to UCSD PASCAL and micro users!" (*80/02/23*)

Earl N. Taver, 195 Varick Rd., Newton, MA 02168: "Have just heard that Hewlett Packard will have PASCAL for HP1000 systems in a few months. Will send info as I get it." (*80/04/01*)

Dr. Richard Yensen, 2403 Talbot Road, Baltimore, MD 21216: "LOVE screen interactive features of UCSD Pascal. We need an interchange format for screen control on different CRT terminals." (*80/05/06*)
aptitudes...in maintenance software systems development and PASCAL applications programming."

Medtronic, Inc. 3055 Old Highway Eight, P.O. Box 1453, Minneapolis, MN 55440 "has a position that recognizes your BSBE, and 6-8 years experience with PASCAL-based computer simulation..." (*80/03/24#)

MTS Systems Corp. P.O. Box 24012, Minneapolis, MN 55424 is looking for a software development engineer for products "based upon latest microprocessor technology. PASCAL and assembly language will be used for implementation." (*80/03/10#)

The New York State Legislature, 250 Broadway - 25th Floor, New York NY 10007 wants a demographer, cartographer, and junior programmers. All applicants should have practical computer programming experience in FORTRAN, COBOL, or PASCAL. (*80/03/10#)

Northern Telecom, P.O. Box 1222, Minneapolis, MN 55440 is looking for a senior programmer/analyst with "high-level programming language (PASCAL, COBOL, BASIC) and compiler writing." (*80/03/24#)

Texas Instruments, P.O. Box 401628, Dallas, TX 75240, has openings in Dallas and Lewisville, Texas, to work "with real-time software applications for min/micro computer based systems and on distributed computer architectures and uni-processor systems." One of the languages: Pascal.

(* Andy Mickel passed on to me the following Want Ad, which appeared in the March 1980 issue of the Pug Press, published by Maryanne Johnson of Excelsior, MN 55331. It is offered here, verbatim, without further comment... *)

WANTED - Small FOG stud to breed with the Classiest Bitch in Town. Stud must be experienced yet gentle, loving, and discreet. Contact Ron or Marlys Hampe (612)-890-4141

MANUFACTURERS' ADVERTISMENTS:

(* A lot of these advertisements appear in several publications; this list is gleaned from a "spot check" of several months' worth of magazines and trade journals. Where a product description is much more detailed than the information given here, a reference is provided. *)

Associated Computer Industries, Inc. 17751 Sky Park East, Suite C, Irvine, CA 92714, announced a Pascal Video terminal for use with UCS Pascal. It accommodates several international languages character displays by internal switch changes, with no optional ROM required. They also sell the ACT-90 Pascal Professional Performance Computer, based on the Western Digital Microengine. Includes the UCS Pascal operating system, and business software: General Ledger, Accounts Payable, Accounts Receivable, Payroll, and Order Entry Inventory.

 Hewlett-Packard Data Systems Division, Dept. 370, 1100 Wolfe Road, Cupertino, CA 95014 offers Pascal for the HP/1000 computer; it has added double-word integer, double-precision data types, random access I/O, and external FORTRAN and assembly language capability.

Intel Corporation of Santa Clara now has Pascal for its Intellec development systems, as reported in the Intel Preview of February 1980. It "encompasses the full standard...as defined in Pascal User Manual and Report by Jensen and Wirth", and "...offers several more extensions to the UCS Standard." The blurb also notes, "The UCS Pascal Implementation has become the industry standard and was the first such implementation of this relatively new programming language." (* The person who sent me this noted, in the margin, "111". I agree. *)

Meta Tech, 8572-1 Via Mallorea, La Jolla, CA 92037 advertises Pascal/MFT, a compiler running under CPM in 32K bytes or more. Compiles a subset of Pascal into ROMable 8080/Z80 code. Object code costs $100, source code costs you OMs $5000.

North Star, 1440 Fourth St., Berkeley, CA 94710, advertises Pascal for its Horizon system.

Oregon Software, 2340 S.W. Canyon Road, Portland, Oregon 97201 announced OMSI Pascal V1.2 with symbolic debugger and profiler, for any RSTS/E, RT-11, RSX-11, or IAS operating system. (* Computerworld 80/01/28#)

Rational Data Systems, 245 W. 55th St., New York, NY 10019 has Pascal for Data General computers, and also puts out a small Pascal Newsletter. (* And, in my opinion, it looks very nice *)

Renaissance Systems, Inc., Suite M, 11760 Sorrento Valley Rd., San Diego, CA 92121 offers Proff and Forml, word processing support programs for formatting and printing text files and aiding in document generation. Written in UCS Pascal, the combination costs $500. Documentation costs $25. (* Computerworld 80/01/14 p. 50 *)

SoftTech Microsystems, 9494 Black Mountain Road, San Diego, CA 92126, offers UCS Pascal "with full documentation and support."

Valley Software Inc., 390-6400 Roberts Street, Burnaby, B.C. Canada V5G 4P2 is a systems/design, programming and consulting service offering Pascal compilers for DEC and Data General.

NEWSLETTERS & ARTICLES:

Brown University Computer Center has arranged to lease a new PASCAL compiler developed at the University of Waterloo; it is the PASCAL described in the British Standards Institute DMS/14/3 Working Draft/3...it offers extended I/O capabilities to allow convenient access to CMS files. (* March 1980 *)

The Institute for Information Systems, Mail Code C-021, University of California at San Diego, La Jolla, CA 92093 is publishing newsletters describing the UCS Pascal System.

Mr. Jim McCord sends a "UCSD Pascal Hobby Newsletter #1." (* Sorry, I have no address on this; could someone out there please provide it? *)
The University of Michigan Computing Center presented a short course on Pascal this April. In the blurb, the newsletter states that..."Pascal offers significant advantages over other languages for general purpose programming." (*80/03/19*)

(* Ah-ha! Here's the article that answers just about all of the "can I get a version of Pascal for my [fill-in-the-blank] microcomputer?" questions. *)

Mini-Micro Systems April 1980 Issue has a lengthy article (pp. 89-110) entitled "High-level languages for microcomputers", by Mokurai Cherlin. Along with the article is a table of microcomputer high-level language suppliers; there are over 40 suppliers of Pascal for fifteen different chips.

The Northwestern University newsletter announced the arrival of the Pascal Release 3 compiler for the Cyber, with compiler options for selecting run-time tests and post-mortem dumps; and defining file buffer and central memory sizes. (*April 1980*)

The University of Southern California is forming a Users Group for PASCAL and ALGOL users. (*February 1980*)

Rick Shaw, Editor
Pascal News
Digital Equipment Corporation
Atlanta, Georgia

Mr. Shaw:

Enclosed is a copy of "A Pascal Bibliography (June 1980)". Although it excludes references to articles on Pascal appearing in magazines such as BYTE and Datamation, it may be of some interest to your readers. (* See Page 12 -ed. *)

If anyone wishes to inform me of errors or omitted articles, I would be grateful to hear from him.

Respectfully,

David V. Notiat
Department of Computer Science
North Carolina State University
Raleigh, North Carolina 27650
BOOKS ABOUT PASCAL

(* This is a complete listing of all known books about Pascal *)


Hartmann, A. C., A Concurrent Pascal Compiler for Microcomputers, Springer-Verlag Lecture Notes in Computer Science, No. 59, 1979, 365 pages, $12.95.


Schneider, G. M., Weingart, S. W. and D. M. Perlman, An Introduction to Programming and Problem Solving with Pascal, Wiley and Sons, 1979, 394 pages.

Webster, C. A. G., Introduction to Pascal, Heyden, 1976, 152 pages, $11.00.


ARTICLES ABOUT PASCAL

(* These articles have appeared since the preparation of #17. *)


Byrnes, John L., "NPS-Pascal: A Pascal Implementation for Microprocessor-Based Computer Systems", Naval Postgraduate School, June 1979, 293 pages, NTIS Report AD-A871972/4HC.


Sites, Richard L. and Daniel F. Perkins, "Universal P-Code Definition, Version (5.3)", Univ. of California at San Diego Dept. of Electrical Engineering, July 1979, 45 pages, UCSD/CS-79/417, NTIS PB-236777/4HC.


A PASCAL BIBLIOGRAPHY
(June, 1980)

David V. Hoffat
North Carolina State University
Fayetteville, North Carolina


[16] L. V. Atkinson, "Know the State You Are In", Pascal Newsletter, 13, 66-69 (December 1978)


[24] M. S. Ball, Pascal 1102: An Implementation of the
Pascal Language for UNIVAC 1100 Series Computers, Naval Ocean Systems Center, San Diego (July 1978)


[38] C. Bishop, "Pascal: Standards and Extensions", Pascal News, 11, 54-56 (February 1978)


[67] J. L. Byrnes, MPS-Pascal: A Pascal Implementation for Microprocessor Based Computer Systems, Naval Postgraduate School, Monterey, California (1979)


[83] D. Corner, "MAP: A Pascal Macro Preprocessor for Large Program Development", Software--Practice and
Experience, 2, 203-209 (1979)


[97] F. Edwards, "Is Pascal a Logical Subset of Algol 68 or Not?", SIGPLAN Notices, 12, 6, 184-191 (1977)


[103] W. Findlay, "The Performance of Pascal Programs on the MULTIM", Report No. 5, Computing Department, University of Glasgow, Scotland (July 1974)


[139] T. Irish, "What to do After a While... Longer",
Pascal News, 13, 65-66 (December 1978)

[140] K. Ishihata and T. Hikita, "Bootstrap Pascal Using a Trunk", Department of Information Science, Faculty of Science, University of Tokyo (1976)


[166] G. Lecarne, "Is Algol 68 a Logical Subset of Pascal or Not?", SIGPLAN Notices, 12, 6, 33-35 (1977)


[170] L. A. Liddiard, "Yet Another Look at Code Generation for Pascal on CUC 6000 and Cyber Machines", Pascal Newsletter, 7, 17-23 (February 1977)


[175] D. C. Luckham and N. Suzuki, "Verification of Array, Record, and Pointer Operations in Pascal", ACM TOPS, 1, 2, 226-244 (1979)


[185] A. Mickel, Pascal Newsletter, University of Minnesota Computer Center, Minneapolis: No. 5 (September 1976), No. 6 (November 1976), No. 7 (February 1977), No. 8 (May 1977), Pascal News (change of name): No. 9 and 10 (September 1977), No. 11 (February 1978), No. 12 (June 1978), No. 13 (November 1978), No. 14 (January 1979), No. 15 (September 1979), No. 16 (October 1979) (See also G. Richmond and R. Shaw)


[212] W. C. Price, "What is a Textfile?", *Pascal News*, 9, 6, 42-42 (September 1977)


[216] B. W. Ravenel, "Will Pascal be the Next Standard Language?", *COMPCON 79 Digest of Papers*, IEEE, 144-146 (1979)


[219] G. Richmond (ed.), *Pascal Newsletter*, University of Colorado Computing Center, Boulder: No. 1 (January 1974); *SIGPLAN Notices*, 6, 3, 21-26 (1974); No. 2 (May 1974); *SIGPLAN Notices*, 5, 11, 11-17 (1974); No. 3 (February 1975); *SIGPLAN Notices*, 1, 2, 33-48 (1976); No. 4 (July 1976) (See also A. Nickl and R. Shaw)

Evaluation then Conditional Expressions", *Pascal News*, 13, 63-65 (December 1978)


[228] A. H. J. Sale, "A Note on Scope, One-Pass Compilers, and Pascal", *Australian Computer Sciences Communications*, 1, 1, 80-82 (1979)


[233] A. Sale, "General Thoughts on Pascal Arising out of Correspondence Between Southampton and Tasmania", *Pascal Newsletter*, 6, 45-47 (November 1976)


[248] E. Shaw (ed.), *Pascal News*, Digital Equipment Corp., Atlanta, Georgia: No. 17 (March 1980), No. 18 (May 1980) (See also A. Nickle and G. Richmond)


[289] W. Wickman, "Pascal is a Natural", IEEE Spectrum, (March 1979)
[294] N. Wirth, Pascal-S: A Subset and its Implementation, Berichte Nr. 12, Institut fur Informatik, Eidgenossische Technische Hochschule, Zurich, Switzerland, 1975
[298] N. Wirth, On "Pascal", Code Generation, and the CDC 6000 Computer, STAN-CS-72-257, Computer Science Department, Stanford University, Stanford, 28 (1972)
That way, they could have done some editing and had a compiler look at the examples—a good way to eliminate errors. (In fact, Kernighan and Plauger used this technique in "Software Tools" (McGraw-Hill), wherein RATS FOR was presented.)

Despite the above, PWS is not a useless book. I found the section treating top-down techniques to be useful. PWS describes other approaches to problem definition/solution and explains why they fail so often. The authors lay out in detail the process of successive refinement. This is clear and to the point. The bibliography contains the standard references to Dijkstra, etc., as well as several less well known sources. The Programmers Proverbs are worth reading and knowing. They are presented with explanations of why they are important.

As for program structure, I must say that in Pascal, especially, data representation is an important part of making programs comprehensible to the human mind. And making programs comprehensible (and correct) is what programming style is all about. Sets, subroutines, and record types are simply not treated.

There are a few nibbling syntax errors. On page 119, for example, a $ is omitted in a procedure declaration. This is curious, and I mention it only because parts of the book appear to have been printed by a Decwriter, implying the text was machine-readable. Why not all of it?
Backissues of Pascal News(letter) from Time Zero  - Andy Mickel  80/07/11.

Pascal Newsletter was started by George Richmond at the University of Colorado Computing Center in early 1974 primarily to spread information about the distribution of the CDC Pascal compiler and to answer questions about other issues. He edited issues 1 through 4. In 1976 Pascal User's Group assumed control of Pascal Newsletter. I changed the name to Pascal News with issue 9. Below are some facts about issues 1 through 16.

<table>
<thead>
<tr>
<th>Date</th>
<th>Issue</th>
<th>pages (numbered)</th>
<th>Estimated printed copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan 1974</td>
<td>Pascal Newsletter #1</td>
<td>8 (8)</td>
<td>2000+SIGPLAN Notices 1974 Mar</td>
</tr>
<tr>
<td>May 1974</td>
<td>Pascal Newsletter #2</td>
<td>18 (18)</td>
<td>2500+SIGPLAN Notices 1974 Nov</td>
</tr>
<tr>
<td>Feb 1975</td>
<td>Pascal Newsletter #3</td>
<td>19 (19)</td>
<td>4000+SIGPLAN Notices 1974 Feb</td>
</tr>
<tr>
<td>Aug 1976</td>
<td>Pascal Newsletter #4</td>
<td>103 (103)</td>
<td>500+230 sent by PUG</td>
</tr>
<tr>
<td>Sep 1976</td>
<td>Pascal Newsletter #5</td>
<td>124 (65)</td>
<td>1150+350 UK</td>
</tr>
<tr>
<td>Nov 1976</td>
<td>Pascal Newsletter #6</td>
<td>180 (91)</td>
<td>1150+350 UK</td>
</tr>
<tr>
<td>Feb 1977</td>
<td>Pascal Newsletter #7</td>
<td>90 (45)</td>
<td>1150+350 UK</td>
</tr>
<tr>
<td>May 1977</td>
<td>Pascal Newsletter #8</td>
<td>128 (65)</td>
<td>1150+450 UK</td>
</tr>
<tr>
<td>Sep 1977</td>
<td>Pascal News #9/10(combined)</td>
<td>220 (113)</td>
<td>3500+600 UK+150 AUS</td>
</tr>
<tr>
<td>Feb 1978</td>
<td>Pascal News #11</td>
<td>202 (105)</td>
<td>3500+600 UK+150 AUS</td>
</tr>
<tr>
<td>Jun 1978</td>
<td>Pascal News #12</td>
<td>175 (69)</td>
<td>3500+600 UK+150 AUS</td>
</tr>
<tr>
<td>Dec 1978</td>
<td>Pascal News #13</td>
<td>239 (123)</td>
<td>4000+750 UK+250 AUS</td>
</tr>
<tr>
<td>Jan 1979</td>
<td>Pascal News #14</td>
<td>61 (61)</td>
<td>4100+750 UK+250 AUS</td>
</tr>
<tr>
<td>Sep 1979</td>
<td>Pascal News #15</td>
<td>247 (125)</td>
<td>4000+750 UK+250 AUS</td>
</tr>
<tr>
<td>Oct 1979</td>
<td>Pascal News #16</td>
<td>305 (135)</td>
<td>4000+750 UK+250 AUS</td>
</tr>
</tbody>
</table>

At PUG(USA) there are approximately 700 copies of 9-12 and 1100 copies of 13-16 left.

Review of Pascal News 13, 14, 15, and 16.  - Andy Mickel 80/07/11.

I would like to urge all new PUG members to consider obtaining backissues 13-16 so that you will be better oriented to events in our recent past.

To describe the highlights: #13 and #15 are the meaty issues. #13 contains the most recent, complete summary of all Pascal compilers to present. The articles in #13 are mostly centered on a lively discussion of control structures. #15 describes a lot of standards activity and the resolution of the future of Pascal and PUG.

#14 is completely devoted to Working Draft 3 of the Pascal Standard, and #16 is completely devoted to a Validation Suite of more than 300 Pascal programs.


Editor’s Contribution: Thanks to those people at the University of Minnesota who have written Pascal News the shadow of their smile, PORTTRAN - The End at Last? Recent events: Employment opportunity, Concurrent Pascal, NASA and the Galileo Project, Conventionalized Extensions, Standards, Pascal Machines, Pascal Usage, Explosion in Industry Literature. Pascal User's Group / Pascal News status: why we are behind.

Here and There: News from Pascaleras; a very large Pascal in the News; another Pascal T-shirt; Pascal in Teaching; Books and Articles; Conference reports: French ACFXT Pascal Group, Australian Computer Science Conference, SIGPLAN ACM meeting, UCSD Pascal Workshop. A Review of Pascal News 9/10, 11, and 12. Roster Increment 78/04/22 - 10/31.


Articles:
- "Moving a Large Pascal Program from an LSI-11 to a Cray-1" - Richard L. Sites
- [A 2600-line Pascal program was moved between 2 machines whose CPU speed ratio is 150 to 1. The task proved easy and 6 portability problems are outlined. Lack of adherence to standards and incompatibilities in the run-time environment were the major areas of difficulty.]
- "On the Article 'What to do After a While'" - Roy A. Wilinker
- [An examination of a table search algorithm is made with respect to considerations of "psychological set," "proving programs correct," "the spirit of Pascal," and "efficiency." Conditional evaluation of Boolean expressions as advocated in the original paper is not necessarily the solution.]
- "A Resolution of the Boolean Expression-Evaluation Question or If Not Partial Evaluation Then Conditional Expressions" - Morris W. Roberts and Robert N. MacDonald
- [The language features of case expression, value block and the conditional expression are recommended as additions to Pascal taken from the precedents of ALGOL-60 and ALGOL-W. An analysis of several control structure constructs is given.]
- "What to do After a While .. Longer" - T.M.N. Irish
- [A thorough reply to Mullins and Barron's article "What to do After a While" arguing against conditional Boolean expression evaluation. He says we should not 1) write programs that rely on ill-defined factors, side-effects of functions, or undefined values, 2) depend on implementors to let us get away with them, or 3) complain if implementors use any means they can devise to prevent us getting away with them.]
- "Know the State You Are In" - Laurence V. Atkinson
- [A number of recent articles have highlighted problems with multiple exit loops in Pascal. Many of these problems disappear when a loop is controlled by a user-defined scalar. The state transition technique is applicable to a number of programming situations and to multi-exit loops in particular.]

Open Forum:
- 78/05/25 Sam Calvin to Andy Mickel: [Department of Defense Dependents schools use of Pascal in Math programs to teach K-12 students with personal instruction]
- 78/06/08 Dave Rasmussen to Andy Mickel: [Building Automation Systems process control language using Pascal, at Johnson Controls In Milwaukee]
- 78/04/24 C. Edward Reid to Andy Mickel: [corrections to letter of 78/03/16 in PN #12 p47]
- 78/12/01 Andy Mickel to PUG members: [The future of PUG and Pascal News; turning the editorship over to someone else. A proposed constitution]
- 78/07/17 Charles L. Hethcoat III to Andy Mickel: [The reference to "Implications of Structured Programming for Machine Architecture" by Andrew Tanenbaum in CACM describing EM-1 a compact instruction machine.]
- 78/07/21 C. Edward Reid to Andy Mickel: [Pointing attention to Dijkstra's article "DOD-1: The Summing Up" in SIGPLAN Notices and highlighting shortcomings]
- 78/07/29 Ralph D. Jeffords to Andy Mickel: [Announcing the construction of 2 software tools in Pascal: LEXGEN and LALRI for Syntax Parsing and Generating.]

Editor's Contribution: A special issue devoted to the Draft Pascal Standard. Notes that Pascal the language and its development have been unique. The appropriateness of letting Europeans standardize a language with European origins.

The BSI / ISO Working Draft of Standard Pascal by the BSI DPS/13/4 Working Group. Letter, Covering Note and Commentary by Tony Addyman; The Draft (6 sections + index); Related Documents: A history, members of DPS/13/4 and the ISO proposal.

Pascal News #15, September, 1979, Pascal User's Group, University of Minnesota Computer Center, 247 pages (125 numbered pages), edited by Andy Mickel.


Here and There: Tidbits (news from Pascalers), a very large Pascal in the News. Ada, Books and Articles including a Textbook survey, Conference and Seminars (4 industry Seminars to be given on Pascal), Announcements for ACM 79 and IFIP 80 reports on the DECUS Pascal SIG ; Pascal session at ACH 78. PUG Finances 77-78; Roaster Increment to 79/05/14.


Articles:
"A Contribution to Minimal Subranges"
- Laurence V. Atkinson

"Enumerated and subrange types are two of the most important features of Pascal. Their contribution to transparency, security and efficiency is often not fully appreciated. Their under-utilization is one of the (many!) features I repeatedly criticize when reviewing Pascal books. Minimal subranging is desirable in Pascal. One benefit of a state transition approach to dynamic processes, is that minimal subranging can be achieved."

"A Note on Scope, One-Pass Compilers, and Pascal"
- Arthur Sale

"The scope rules set out in section 2 and now incorporated into the draft Pascal Standard are sufficient to permit even one-pass compilers to reject incorrect programs. The suggested algorithm adds an overhead at every defining occurrence, but since uses exceed definitions in general it may not be too expensive in time to implement. In any case, what price can be put on correctness?"

"Pascal-i = Interactive, Conversational Pascal-5"
- Richard Cichelli

"Pascal-i is a version of the Wirth Pascal-S system designed to interact with the terminal user. The system contains a compiler, interpreter, text editor, formatter, and a run-time debugging system. A description of commands and a terminal session are given."

"Tracing the Heap"
- Steve Schach

[The package HEAPTRACE outlined in this paper aids the user to debug his programs by providing information as to the contents of the records on the heap. Each field is named, and its value is given in what might be termed "high-level format".]

Implementation Notes:

General Information, Implementors Group Report, Checklist, Portable Pascal:
Pascal-P, Pascal P4--Bug Reports, Pascal Trunk, Pascal J; Pascal Variants:
"Why Use Structured Formatting"
- John Crider

["Structured Formatting" is a technique for prettyprinting Pascal programs. It is based on a single indented display pattern which is used to display almost all of the structured statements in a Pascal program.]

Open Forum:
79/03/30 David Barron to Andy Mickel: [Thoughts on the future of PUG prompted by Open Letter in #13. PUG has succeeded beyond all reasonable expectation because it has been informal and unconventional.]
79/03/12 Paul Brainerd to Andy Mickel: [Understands the time to produce Pascal News and we should pick a new editor carefully and perhaps be realistic about price.]
79/03/19 John Earl Crider to Andy Mickel: [Pascal News has become an impressive journal that ... I am sure serves most other PUG members as their major link to Pascal developments.]
79/03/19 John Eisenberg to Andy Mickel: [The Bald Organization--An Anti-Constitution For Pascal User's Group]
79/05/01 Jim Miner to Friends of PUG: [Save the PUG! What is PUG? On the Proposed Constitution. Where Now, FUG?]
79/05/12 Rich Stevens to Jim Miner: [I agree with Save the PUG. Would rather see a smaller, more frequent publication.]
79/05/18 Arthur Sale to Jim Miner: [I agree with Save the PUG. Constitution would effectively eliminate international cooperation by ignoring it.]
79/05/20 David Barron to PUG membership: [I agree with Save the PUG. The only real function of PUG is to publish Pascal News.]
79/05/31 Gregg Marshall to Andy Mickel: [I oppose any movements which advocate dissolution, or radical change from the current editorial policies.]
79/06/30 Bill Heidbrecht to Andy Mickel: [PUG must be kept alive, independent, and international--it has outlived its usefulness.]
78/09/30 Tom King to Andy Mickel: [Use of Pascal on an AM-100 system in Winneumucca, Nevada with varied applications.]
79/06/02 John Eisenberg to Andy Mickel: [Arguments over the use of Pascal and Pascal Standards and extensions.]
79/10/16 Robert Cailliau to Andy Mickel: [Comments on Pascal News #12 standards and extensions.]
79/10/22 C. Roads to Andy Mickel: [Pascal in Music applications in the Computer Music journal.]
79/11/07 Laurent O. Gelinder to Andy Mickel: [Applications on a large file processor and intelligent terminals network.]
79/11/08 Eugene Myia to Andy Mickel: [Jet Propulsion Labs and Pascal on their 300 computers: the Deep Space Network and need for validation programs.]
79/11/27 Paul Lebreton to Andy Mickel: [News on the Motorola 68000 and Pascal and bus standards and other hardware conventions.]
79/11/21 Sergei Pokrovsky to Andy Mickel: [Use of a double-variant node in Pascal used to create a syntax for graph structures.]
79/03/26 Bill Marshall to Andy Mickel: [Deviations in 4 compilers for TRUNC and ROUND]
79/02/09 Curt Hill to Andy Mickel: [Pascal at the University of Nebraska: good report on the Stanford 360/370 compiler.]
79/03/08 James Cameron to Andy Mickel: [The problems of extensions might be solved by also providing a superset language "PascalII!"
79/03/13 Roger Gulbranson to Andy Mickel: [Reply to Richard Cichelli's claim that complex numbers are easy to create in Pascal. Probably need an Operator declaration.]
79/04/30 B. J. Smith to Andy Mickel: [The production of various Software Tools in Pascal by Interactive Technology INC. including a DBMS and business applications.]
79/07/20 Peter Humble to Andy Mickel: [Need for conformant arrays in Pascal for numerical applications.]
79/06/05 George Richmond to Andy Mickel: [Pascal at Storage Technology Corp. Errors in the Pascal-P compiler.]
79/06/02 Bob Schor to FUG: [Pascal at Rockefeller University and on FDP-II's]
79/06/29 Jack Dodds to Tony Addyman: [The need for conformant arrays in Pascal for the use of libraries and a better definition of EXTERNAL]
79/09/20 Andy Mickel to Ken Bowles: [Pascal-P is public-domain software and UCSD Pascal is based on Pascal-P, yet Improper modification history and credit is made.]

Pascal Standards:
79/03/19 News Release by CBEMA on behalf of ANSI regarding the solicitation of public comments on the ISO draft standard for Pascal.
79/08/31 Experiences at the Boulder, Colorado meeting of IEEE/X3J9 committee by Andy Mickel. More politics.-

Validation Suite:
Announcement by Arthur Sale of the distribution centers and prices for the forthcoming Pascal Validation Suite.

Implementation Notes:

Pascal News #16, October, 1979, Pascal User's Group, University of Minnesota Computer Center, 305 pages (155 numbered pages), edited by Andy Mickel.

Editor's Contribution: A special issue devoted to the Pascal Validation Suite. Rick Shaw is a new editor of Pascal News. Thanks to everyone. How we put together an issue of Pascal News. Final thoughts on the PUG phenomenon. Greetings from the new editor and predictions of the next two issues.

PUG FINANCES 1978-1979 (Actually through 79/12/12 just before transfer to Atlanta)

Here are the details for PUG(USA)'s finances for the 78-79 academic year. We have not included PUG(UK) because they will report separately. PUG(AUS) never has reported.

PUG(USA) Summary of Accounts:

Income:
- 196.53 1977-78 Surplus
- 334.94 1976-77 Surplus (forgot to include on 77-78 accounting!)
- 197.20 Interest on Bank Account
- 87.30 Contributions
- 510.00 Sale of 513 sets of backissues (9..12) @ $10
- 66.00 Sale of 33 miscellaneous backissues (5..8) @ $2
- 132.00 Sale of 44 miscellaneous backissues (9..14) @ $3
- 2500.00 625 subscriptions @ $4
- 10950.00 10950 subscriptions @ $10

Total income: $20,200.92

Expenses:
- 181.00 People who still owe us money (bounced checks)
- 104.91 Missing meeting notices
- 319.45 Advance printing #14 - 200 copies
- 1341.00 Printing #14 - 3000 copies
- 3538.92 Printing #15 - 4000 copies
- 6080.00 Printing #16 - 4000 copies
- 122.86 Postage due from returned issues
- 414.76 Postage #13
- 307.96 Postage #14
- 534.65 Postage #15
- 629.02 Postage #16
- 34.27 Miscellaneous photocopying costs, postage
- 50.48 UPS shipping of the files to Atlanta from Minneapolis
- 935.24 PUG(UK) 1977-78 rebate
- 784.90 Reprinting #12 - 500 copies

Total expenditure: $19,593.97

$19,593.97 Total income.

$20,200.92 Total expenditure. Excess expenditure = $606.95

An attempt to assess the financial health of PUG:

Assets: $29,886.86
- Bank Account 1930.43
- Computer Center Account 7000.00
- Cash sent to Atlanta to start up 4448.00
- Face value of 3566 backissues 6858.00
- on hand =cost to print 6858.00
- 80-81 subscriptions collected 1808.00
- 82 subscriptions collected 830.00

$16,363.79 Total assets.

$606.95 Total liabilities.

I claim we didn't do too bad. Since 79/12/12 we have spent almost all of the remaining cash here in Minneapolis on reprinting backissues 9..14. These details will be reported with the 79-80 report by Rick.

Andy Mickel 80/06/24.

Computer Systems Represented by the PUG Membership 1976-1979:

Here is a list of the computer systems listed on All-Purpose Coupons by the 4676 different members of Pascal User's Group from 76/03/03 through 79/11/01 (the last date for which I processed PUG memberships). Duplicate listings from the same people on different (renewal, change of address, etc.) coupons were eliminated.

Unfortunately I don't know all these computer systems so I may have many misplaced (alphabetically by manufacturer); check through the whole list if you are looking for a system in particular.

As PUG member A. J. Sutton so aptly stated on his 78/10/15 coupon: "cheers, but what does this [computer system(s)] mean? I owned? Operated? Programmed? Designed? Delivered? Desired?" I guess I meant using, so take these figures with a grain of salt!

Andy Mickel 80/06/24.

(Note: the notation (+n) indicates additional quantity for micros under a different name.)
<table>
<thead>
<tr>
<th>Company</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hewlett-Packard</td>
<td>1000</td>
</tr>
<tr>
<td>Hewlett-Packard</td>
<td>2000/2100</td>
</tr>
<tr>
<td>Hewlett-Packard</td>
<td>2200</td>
</tr>
<tr>
<td>Hewlett-Packard</td>
<td>3000</td>
</tr>
<tr>
<td>Hitachi</td>
<td>8000</td>
</tr>
<tr>
<td>Honeywell</td>
<td>H316</td>
</tr>
<tr>
<td>Honeywell Level 6</td>
<td>66/68</td>
</tr>
<tr>
<td>IBM Series 1</td>
<td>1</td>
</tr>
<tr>
<td>IBM System 3</td>
<td>7</td>
</tr>
<tr>
<td>IBM System 32/34</td>
<td>14</td>
</tr>
<tr>
<td>IBM 1130</td>
<td>430</td>
</tr>
<tr>
<td>IBM System 360/370</td>
<td>36</td>
</tr>
<tr>
<td>IBM 3030</td>
<td>1</td>
</tr>
<tr>
<td>IBM 4330</td>
<td>2</td>
</tr>
<tr>
<td>ICL 1900</td>
<td>23</td>
</tr>
<tr>
<td>ICL 2900</td>
<td>2</td>
</tr>
<tr>
<td>ILLIAC IV</td>
<td>1</td>
</tr>
<tr>
<td>IMSAI VDP</td>
<td>6</td>
</tr>
<tr>
<td>IMSAI 8080/8085</td>
<td>31</td>
</tr>
<tr>
<td>Intel 8080 (+73)</td>
<td>16</td>
</tr>
<tr>
<td>Intel 8085 (+5)</td>
<td>18</td>
</tr>
<tr>
<td>Intel 8086</td>
<td>16</td>
</tr>
<tr>
<td>Itel (National) AS 456</td>
<td>2</td>
</tr>
<tr>
<td>Ithaca Audio</td>
<td>1</td>
</tr>
<tr>
<td>ITT 1652</td>
<td>1</td>
</tr>
<tr>
<td>ITT 2020</td>
<td>1</td>
</tr>
<tr>
<td>Jacquard J-100</td>
<td>8</td>
</tr>
<tr>
<td>KIM-1</td>
<td>1</td>
</tr>
<tr>
<td>LEC-16</td>
<td>2</td>
</tr>
<tr>
<td>Lekheed Sue</td>
<td>3</td>
</tr>
<tr>
<td>Manchester MU-5</td>
<td>1</td>
</tr>
<tr>
<td>Marinchip 9900</td>
<td>1</td>
</tr>
<tr>
<td>MDS-800</td>
<td>1</td>
</tr>
<tr>
<td>MEMBRAIN</td>
<td>2</td>
</tr>
<tr>
<td>Microdata 32/5</td>
<td>2</td>
</tr>
<tr>
<td>Microdata 1630</td>
<td>2</td>
</tr>
<tr>
<td>MITS Altair 6800</td>
<td>17</td>
</tr>
<tr>
<td>MITS Altair Z-80</td>
<td>1</td>
</tr>
<tr>
<td>Mitsubishi MELCOM 7700</td>
<td>2</td>
</tr>
<tr>
<td>MIM Linolex</td>
<td>4</td>
</tr>
<tr>
<td>MODCOMP II</td>
<td>15</td>
</tr>
<tr>
<td>MODCOMP IV</td>
<td>9</td>
</tr>
<tr>
<td>Mostek 6502 (+44)</td>
<td>14</td>
</tr>
<tr>
<td>Motorola 6800 (+10)</td>
<td>67</td>
</tr>
<tr>
<td>Motorola 6809</td>
<td>8</td>
</tr>
<tr>
<td>Motorola 68000</td>
<td>10</td>
</tr>
<tr>
<td>Nanodata NM-1</td>
<td>4</td>
</tr>
<tr>
<td>National Semiconductor 5-400</td>
<td>2</td>
</tr>
<tr>
<td>National Semiconductor 2900</td>
<td>4</td>
</tr>
<tr>
<td>National Semiconductor PACE</td>
<td>16</td>
</tr>
<tr>
<td>NCR Century</td>
<td>10</td>
</tr>
<tr>
<td>NCR 8000</td>
<td>1</td>
</tr>
<tr>
<td>NEC-900</td>
<td>11</td>
</tr>
<tr>
<td>NEC-3200</td>
<td>1</td>
</tr>
<tr>
<td>Norsk Data NORD-10</td>
<td>14</td>
</tr>
<tr>
<td>North Star Horizon (Z-80)</td>
<td>19</td>
</tr>
<tr>
<td>Northwest Micro 85/P</td>
<td>5</td>
</tr>
<tr>
<td>Odell System 85</td>
<td>1</td>
</tr>
<tr>
<td>Ohio Scientific Challenger</td>
<td>11</td>
</tr>
<tr>
<td>Ontel GP-1</td>
<td>2</td>
</tr>
<tr>
<td>PDS-4</td>
<td>1</td>
</tr>
<tr>
<td>Pertec PCC XL80</td>
<td>8</td>
</tr>
<tr>
<td>Pertec PCC 200</td>
<td>4</td>
</tr>
<tr>
<td>Perkin Elmer Interdata 7/16</td>
<td>45</td>
</tr>
<tr>
<td>Perkin Elmer Interdata 7/32</td>
<td>30</td>
</tr>
<tr>
<td>Perkin Elmer Interdata 8/16</td>
<td>1</td>
</tr>
<tr>
<td>Perkin Elmer Interdata 8/32</td>
<td>28</td>
</tr>
<tr>
<td>Perkin Elmer 3200</td>
<td>7</td>
</tr>
<tr>
<td>Polyphorics 88</td>
<td>4</td>
</tr>
<tr>
<td>Prime P-300</td>
<td>11</td>
</tr>
<tr>
<td>Prime P-400</td>
<td>34</td>
</tr>
<tr>
<td>Prime P-500</td>
<td>4</td>
</tr>
<tr>
<td>Processor Technology SOL-20</td>
<td>12</td>
</tr>
<tr>
<td>Quasar 6800</td>
<td>1</td>
</tr>
<tr>
<td>Quotron 8001</td>
<td>20</td>
</tr>
<tr>
<td>Radio Shack TRS-80</td>
<td>1</td>
</tr>
<tr>
<td>RCA 301</td>
<td>5</td>
</tr>
<tr>
<td>RCA 1002</td>
<td>1</td>
</tr>
<tr>
<td>Rockwell 6502</td>
<td>3</td>
</tr>
<tr>
<td>ROLM 1600</td>
<td>1</td>
</tr>
<tr>
<td>RP-16</td>
<td>2</td>
</tr>
<tr>
<td>SBC 80/20</td>
<td>20</td>
</tr>
<tr>
<td>Systems Engineering SEL 32</td>
<td>3</td>
</tr>
<tr>
<td>Systems Engineering SEL 8600</td>
<td>3</td>
</tr>
<tr>
<td>SEMS SOLAR</td>
<td>1</td>
</tr>
<tr>
<td>SEMS T1600</td>
<td>1</td>
</tr>
<tr>
<td>Siemens 4000</td>
<td>5</td>
</tr>
<tr>
<td>Siemens 7000</td>
<td>8</td>
</tr>
<tr>
<td>Singer OP-48</td>
<td>1</td>
</tr>
<tr>
<td>Singer Librascope</td>
<td>1</td>
</tr>
<tr>
<td>Singer System 10</td>
<td>2</td>
</tr>
<tr>
<td>SORD M-222</td>
<td>1</td>
</tr>
<tr>
<td>SPC-16</td>
<td>2</td>
</tr>
<tr>
<td>SPP-16</td>
<td>1</td>
</tr>
<tr>
<td>Sperry SED-175</td>
<td>5</td>
</tr>
<tr>
<td>SWP 6800</td>
<td>53</td>
</tr>
<tr>
<td>Sycor (Northern Telecom) 445</td>
<td>5</td>
</tr>
<tr>
<td>Tandem 16</td>
<td>6</td>
</tr>
<tr>
<td>TDL Z-80</td>
<td>1</td>
</tr>
<tr>
<td>TDS-8 (Z-80)</td>
<td>1</td>
</tr>
<tr>
<td>Tektronix 8002</td>
<td>7</td>
</tr>
<tr>
<td>Telefunken 80</td>
<td>3</td>
</tr>
<tr>
<td>Telefunken TR-440</td>
<td>2</td>
</tr>
<tr>
<td>Terak 8510</td>
<td>67</td>
</tr>
<tr>
<td>Three Rivers PERQ</td>
<td>3</td>
</tr>
<tr>
<td>Texas Instruments 980</td>
<td>10</td>
</tr>
<tr>
<td>Texas Instruments 990</td>
<td>53</td>
</tr>
<tr>
<td>Texas Instruments 9900</td>
<td>19</td>
</tr>
<tr>
<td>Texas Instruments ASC</td>
<td>5</td>
</tr>
<tr>
<td>Texas Instruments DX-10</td>
<td>2</td>
</tr>
<tr>
<td>Time Machine TM-600</td>
<td>1</td>
</tr>
<tr>
<td>Univac 418</td>
<td>32</td>
</tr>
<tr>
<td>Univac 90/9000</td>
<td>156</td>
</tr>
<tr>
<td>Univac 11000</td>
<td>36</td>
</tr>
<tr>
<td>Univac V70/77</td>
<td>3</td>
</tr>
<tr>
<td>Univac VUK-7</td>
<td>3</td>
</tr>
<tr>
<td>Vector Graphics MZ</td>
<td>1</td>
</tr>
</tbody>
</table>

53 unspecified microprocessors
Applications

Corrections for Xref program. Pascal News #17

**************************************** 1) XREF,PAS,1
464   LinesOnPage := LinesPerPage;   MoveToIndx := () (* compress table *);
465   for TblIndx := 0 to HashTblSize = 1 do

************************************************** P XREF,PAS,1
115& OutputSection := listing; scan; OutputSection := idents;
1157 DumpTables; writeln(tty, 'End CrossRef'); writeln(tty, '');

************************************************** 2) XREF,PAS,2

1156   OutputSection := listing; scan; OutputSection := idents;
1157   DumpTables; writeln(tty, 'End CrossRef'); writeln(tty, '');

2 DIFFERENCES FOUND
LP:=DP1:XREF,PAS;1,DP1:XREF,PAS;2

All occurences of ChrCatagory should be changed to ChrCategory.
E. T. H. The program was modified to compile on DEC PDP 11's using the Swedish Compiler.

Scalar types were added using Don Baccus' changes.

The choice of features to be included in the subset now called PASCAL-S was mainly guided by the contents of traditional introductory programming courses. Beyond this it is subject to personal experience, judgement, and prejudice. A firm guideline was provided by the demand that the system must process a strict subset of PASCAL, i.e. that every PASCAL-S program must also be acceptable by the compiler of Standard PASCAL without being subjected to the slightest change. This rule makes it possible for students to switch over to the regular system in later courses without noticing. A language's power and its range of applications largely depend on its data types and associated operators. We may also determine the amount of effort required to master a language. PASCAL-S adheres in this respect largely to the tradition of ALGOL 60. Its primitive data types are the integers, the real numbers, and the Boolean truth values. They are augmented in a most important and crucial way by the type char, representing the available set of printable characters. Omitted from PASCAL are the scalar types and subrange types.

PASCAL-S included only two kinds of data structures:

- the array and the record (without variants). Omitted are the set and the file structure. The exceptions are the two standard textfiles input and output which are declared implicitly (but must be listed in the program heading). A very essential omission is the absence of pointer types and thereby of all dynamic structures. Of course, also all packing options (packed records, N end).

The choice of data types and structures essentially determines the complexity of a processing system. Statement and control structures contribute but little to it. Hence, PASCAL-S has no more than the basic structures (compound, conditional, selective, and repetitive statements). The only omissions are the 'with' and the goto statement. The latter was omitted very deliberately because of the principal use of PASCAL-S in teaching the systematic design of well-structured programs. Procedures and functions are included in their full generality. The only exception is that procedures and functions cannot be used as parameters.

* Computer system:
PASCAL-S was originally installed on the CDC 6600 systems at E.T.H. The program was modified to compile on DEC PDP 11's using the Swedish Compiler.

Scalar types were added using Don Baccus' changes.

The choice of features to be included in the subset now called PASCAL-S was mainly guided by the contents of traditional introductory programming courses. Beyond this it is subject to personal experience, judgement, and prejudice. A firm guideline was provided by the demand that the system must process a strict subset of PASCAL, i.e. that every PASCAL-S program must also be acceptable by the compiler of Standard PASCAL without being subjected to the slightest change. This rule makes it possible for students to switch over to the regular system in later courses without noticing. A language's power and its range of applications largely depend on its data types and associated operators. We may also determine the amount of effort required to master a language. PASCAL-S adheres in this respect largely to the tradition of ALGOL 60. Its primitive data types are the integers, the real numbers, and the Boolean truth values. They are augmented in a most important and crucial way by the type char, representing the available set of printable characters. Omitted from PASCAL are the scalar types and subrange types.

PASCAL-S included only two kinds of data structures:

- the array and the record (without variants). Omitted are the set and the file structure. The exceptions are the two standard textfiles input and output which are declared implicitly (but must be listed in the program heading). A very essential omission is the absence of pointer types and thereby of all dynamic structures. Of course, also all packing options (packed records, N end).

The choice of data types and structures essentially determines the complexity of a processing system. Statement and control structures contribute but little to it. Hence, PASCAL-S has no more than the basic structures (compound, conditional, selective, and repetitive statements). The only omissions are the 'with' and the goto statement. The latter was omitted very deliberately because of the principal use of PASCAL-S in teaching the systematic design of well-structured programs. Procedures and functions are included in their full generality. The only exception is that procedures and functions cannot be used as parameters.

* Computer system:
PASCAL-S was originally installed on the CDC 6600 systems at E.T.H. The program was modified to compile on DEC PDP 11's using the Swedish Compiler.

Scalar types were added using Don Baccus' changes.
procedure nextch { read next character; process line end }

begin
    procedure readsec;
    begin
        while erros < CO do
            while not (k in erros) do k := k + 1;
            writeln(k, ' ', msg[k]);
            erros := erros + 1 end;
        writeln('program incomplete'); errors := 0; abandon;
    end;
    if erros <> 0 then procedure endskip;
    while erros < CO do begin write('-'); erros := erros + 1 end;
    skipflag := false;
    while not eoln(input) do
        uppercase := chr(ord(ch) - ord('.'), if ch = 'E' then readscale;
        while not odd(s) do begin s = s div 2; d := sqr(d); s := (s - 1) div d;
        until s = 0;
    if k > e then writeln('error(21)');
    if k < e then
        if ch = '=' then
            if ch in ['A' .. 'Z'] then
                while not (ch in ['A' .. 'Z']) do
                    ch = 'A' + 1;
                if ch = 'E' then readscale;
                if ch in ['A' .. 'Z'] then
                    ch = 'A' + 1;
                if ch = 'E' then readscale;
                if ch = 'E' then readscale;
                if ch = 'E' then readscale;
                if ch = 'E' then readscale;
                if ch = 'E' then readscale;
        if errpos < 0 then begin invert = false; end;
        if errpos = 0 then
            if errpos < 0 then
                error(1); error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
                error(1);
procedure enterblock;

procedure enterarrayCtp: types; l, h: integer);

--- "t:=

begin

begin

11 CabsC

--- segin

erse

iTS;-a,;;ax

erse

ii'fth

begin

while tab[jJ.name <> id do j := tab[jJ.link;

rconst[cl name, link: 5,

name, link: 5;

error(n); skipflag:= true;

begin

name := x0;

link := t := 1;

obj := x1;

typ := x2;

ref := 0;

normal := true;

lev := 0;

adr := x3

end

end |

{ enter --- }

(1 

procedure enter(x0: alfa; x1: object; x2: types; x3: integer);

begin

l := t + 1;

{| enter standard identifier |

with tab[tJ do

begin

name := x0;

link := t := 1;

obj := x1;

typ := x2;

ref := 0;

normal := true;

lev := 0;

adr := x3

end

end |

end;

end;

procedure enterblock;

begin

if b = max then fatal(2)

else

begin

b := b + 1;

btabD3.last := 0;

btabD3.lastpar := 0 end

end |

{ enterblock };

procedure enterreal(x: real);

begin

if c2 = cmax - 1 then fatal(3)

else

begin

rconst[c2 + 13 := x; c1 := 1;

while rconst[c1] <> do c1 := c1 + 1;

if c1 > c2 then c2 := c1;

end |

{ enterreal };

procedure emit(fct: integer);

begin

begin

if lc = cmax then fatal(6);

with codeCl do begin f := fct;

x := a;

y := b end

lc := lc + 1

end |

emit

procedure emit(fct, bi: integer);

begin

if lc = cmax then fatal(6);

with codeCl do begin f := fct;

y := b end

lc := lc + 1

end |

emit

procedure emit2(fct, ay, bi: integer);
function loc(id: alfa): integer; 
begin 
  loc3D.name := id (sentinel); 
  repeat 
    if tab[id].name = id (sentinel) then break; 
    else begin 
      loc3D.name := id (sentinel); 
      repeat 
        until (i < 0) or (j < 1); 
        if j = 0 then error(0); 
        loc := j; 
      end; 
    end; 
  end; 
end;
procedure constantdeclaration;
881 offset := offset + else
882 end
883
884
type
885 if s0 <> ends then
886 begin
887 if y = semicolon then insymbol else
888 begin
889 error(4)
890
891 if s0 = comma then insymbol end
892 test(ident, endsy, semicolon), fys, 6)
893 end
894 end
895 end![tabl]r3.size := offset; sz := offset;
896 [tabl]r3.size := 0; insymbol: level := level - 1
897 end
898 test(fys, 1, 6)
899 end[typ] 1)
900
901 procedure parameterlist { formal parameter list };
902 var
t0: types;
903 rf, sz, t0: integer;
904 var: boolean;
905 begin
906 insymbol; tp := typ; rf := 0; sz := 0;
907 test(ident, varsy), fys = [paren], 7)
908 while s0 in [ident, varsy] do
909 begin
910 if sy <> varsy then valpar := true
911 else begin
912 begin
913 test(ident, endsy, semicolon)
914 if sy = colon then insymbol: else semicolon; end
915 test(fys, 1, 6)
916 if sy <> ident then error(2)
917 else
918 begin
919 x := loc(id); insymbol;
920 if x < 0 then with tabdo do
921 if obj <> type then error(29)
922 else
923 begin
924 tp := typ; rf := ref;
925 if valpar then sz := addr else sz := 1
926 end
927 test(secretion, parent), [comma, ident + fys, 14)
928 end
929 else error(5)
930 while t < d do
931 begin
932 t := t + 1;
933 with tabdo do
934 begin
935 typ := tp; ref := rf;
936 addr := dx; lev := level;
937 dx := dx + sz
938 end
939 if sy <> parent then
940 begin
941 if sy = semicolon then insymbol
942 else begin
943 test(secretion, colon), fys, 6) and
944 if sy <> ident then error(3)
945 else
946 begin
947 while 0
948 if sy <> parent then
949 begin
950 if sy = semicolon then insymbol
951 else begin
952 test(secretion, colon), fys, 6)
953 end
954 end
955 end
956 procedure constantdeclaration;
957 var
ci: connect;
958 begin
959 insymbol; test(ident3, blockbegsys, 2);
960 while s0 = ident do
961 begin
962 enter(id, constant); insymbol;
963 if sy = col then insymbol
964 else begin
965 test(secretion, comma, ident) + fys, 6)
966 tabcl.typ := c.tp; tabcl.ref := 0;
967 if c.tp = real
968 then begin enter(real(c.x)); tabcl.3adr := ci and
969 else tabcl.3adr := ci
970 testsemicol
971 end[constantdeclaration] 1)
972 procedure typedeclarartion;
973 var
t0: types;
974 rf, sz, t0: integer;
procedure simpleexpression(fs: symset; var x: item);

var
y: item;
op: symbol;

begin

procedure term(fs: symset; var x: item);

var
y: item;
op: symbol;
ts: typset;

procedure factor(fs: symset; var x: item);

var
f: integer;

procedure standstn(integer);

var
Ts: typset;

begin

expression(fs + [parent], x);

end

{ standard function no. n }

begin

expression(fsys + [parent], x);

if sy = lparent then expression(error(9));

if n < 17

begin

expression(fsys + [parent], x);

end

{ round, trunc, sin, cos, sqrt, ln, log2 }
begin { simpleexpression }
if sy in [plus, minus, times, rdiv, idiv, imod, andsy] then begin
    op := sy; insymbol; term(fsys + [plus, minus], x); if x.typ > reals then error(33) else
    if op = minus then if x.typ = reals then emit(66) else emit(66) else
term(fsys + [plus, minus, orsy], x); while sy in [plus, minus, orsy] do
    begin
        op := sy; insymbol; termtpe(fsys + [plus, minus, orsy], x); if op = orsy then
        begin
            if x.typ = reals then emit(66) else emit(66) else
            term(fsys + [plus, minus, orsy], y); if op = orsy then
            begin
                if x.typ <> notyp and (y.typ <> notyp) then error(32); else
                test(fsys, factegeqsys, 6) while ( )
        end
    end
end { simpleexpression }
begin { expression }
simpleexpression(fsys + [becomes, eql, neq, lss, leq, gtr, geq], x); if sy in [becomes, eql, neq, lss, leq, gtr, geq] then begin
    if sy = becomes then begin error(6); op := eql end else
    op := sy; insymbol; simpleexpression(fsys, y); if ftyp in [bools, ints, bools, chars, scalars] and (x.typ = y.typ) and (x.ref = y.ref) then begin
    case op of
        eql: emit(45); neq: emit(46); lss: emit(47); leq: emit(48); gtr: emit(49); geq: emit(50); end else
    begin
        if x.typ = ints then begin x.typ := reals; emit(61, 1) end else
        if y.typ = ints then begin y.typ := reals; emit(61, 0) end else
        if (x.typ = reals) and (y.typ = reals) then begin
            case op of
eql: emit(39); neq: emit(40); lss: emit(41); leq: emit(42); gtr: emit(43); geq: emit(44)
        end else
        begin
            if x.typ <> notyp and (y.typ <> notyp) then error(32); else
            test(fsys, factegeqsys, 6) while ( )
        end
    end
end { expression }

procedure assignment (lv, ad: integer); var x, y: item; f: integer;
    [ tab1, obj in [variable, procedure] ]
begin
    x.typ := tab1.typ; x.ref := tab1.ref; if tab1.normal then f := 0 else f := 1;
    emit(f, lv, ad); if sy in [brack, lparent, period] then emitvar(becomes, equl + fsys, x); x.typ := reals; emit(61)
    begin
        if ftyp in [sttypes] then emit(58) else
        fref <> y.ref then error(46) else
        if x.typ = arrays then emit(25, tab2x.ref.size) else emit(25, tab2x.ref.size)}
procedure casestatement;
  begin
  insymbol; expression(fs, x); y := lc;
  if y = then insymbol else error(14);
  statement(fs, x); y := lc
  end;

end;

procedure fstatement;
  begin
  fstatement l;
  expression(fsys, x);
  if y = then insymbol else error(54);
  statement(fs, x); y := lc
  end;

end;

procedure repeatstatement;
  begin
  repeatstatement i
  end;

end;

procedure onecase;
  begin
  onecase l;
  expression(fsys, x);
  if y = then insymbol else error(54);
  statement(fs, x); y := lc
  end;

end;

procedure whilestatement;
  begin
  while sy in \[semicolonJ
  do
    expression(fsys); y := lc
    stmtbegsys do
      emit(10, c); emit(38)!!!!!;
    end
    emit(13, 2i); emit(13, 1i); emit(38)!!!!!.
  end;

end;

procedure forstatement;
  begin
    forstatement l;
    expression(fsys, x);
    if y = then insymbol else error(54);
    statement(fs, x); y := lc
    end;

end;

procedure caselabel;
  var
  x: item;
  lc, lc2: integer;

begin
  insymbol; expression(fs = Ethensy, dosy, 2); x;

end;

procedure caseLabel;
  var
    x: item;
    i, j, k, lab: integer;
    casetab: array [1 .. cmax] of packed record
      lab, i: index
      end
      end;

begin
  constant(fs, "comma", colon, l);
  if (lab.tp <> x.tp) or (lab.ref <> x.ref) then error(47)
  else
    if y = then fatal(6)
    else
      if i = cmax then fatal(6)
      begin
        i := i + 1;
        k := 0;
        casetab[l].val := lab.1;
        repeat
          k := k + 1
          until casetab[k].val = lab.1;
        if k < i then error(5) (multiple definition)
      end;

end;

procedure onecase;
  begin
  onecase l;
  insymbol; expression(fs = Ethensy, dosy, 3 + fsys, 2);
  if y = becomes

end;

procedure standproc(n: integer);
  var
    x: item;
    lc, lc2: integer;

begin
  insymbol; if y = iden
  then
    begin
      t := loc(10); insymbol;
      if i = 0 then begin cvt := ints; cvr := 0 end
      else begin cvt := nil; cvr := 0 end
      if tab[l].obj = variable
      then
        begin
          cvt := tab[l].typ;
          if not tab[l].normal then error(57)
          else emit(0, tab[l].lev, tab[l].adr);
          if not (cvt in [ints, bools, chars, scalars])
          then error(78)
          end;

        end;

      else begin
        begin
          if (x.typ = reals) and (y.typ = ints)
          then error(72)
          else code[lc2] := lc
        end;

      end;

    end;

end;

begin
  case n of
    1, 2:
    begin ( read )
      if not (sy = dosey then insymbol else error(54))
      then
        begin
          insymbol;
        end;
      if y <> iden then error(2)
      else
        begin
          if (x.typ = reals) and (y.typ = ints)
          then error(72)
          else
            if i < 0
            then
              begin
                if tab[l].obj <> variable
                then error(37)
                else begin
                  end;

        end;

end;

begin
  case n of
    1, 2:
    begin ( read )
      if not (sy = dosey then insymbol else error(54))
      then
        begin
          insymbol;
        end;
      if y <> iden then error(2)
      else
        begin
          if (x.typ = reals) and (y.typ = ints)
          then error(72)
          else
            if i < 0
            then
              begin
                if tab[l].obj <> variable
                then error(37)
                else begin
                  end;

        end;
Notes on system dependent code in Pascal-S and Pascal-I.

by Richard J. Cichelli

Pascal-S had a 'trap label' to recover (just once) from user errors that cause aborts. In Pascal-I, John McGrath, Curt Loughin and I solved similar problems with what we think are cleaner, simpler and more generally useful techniques. We'd like to share them with you here.

Pascal-I: Interactive, conversational Pascal-S. These code fragments from Pascal-I show nearly all of the non-standard and/or system dependent parts of the 7500 line program that is Pascal-I.

The code illustrates how functionality, which must be provided for the system to work in its given environment and obviously cannot be specified in a standard way, can be isolated so that reasonable portability can be obtained.

Of particular note is the method for recovering from timeouts and user aborts. On a user abort, Pascal-I terminates the user initiated action, recovers and accepts the next user command request. Pascal-I also does interactive I/O.

const
  . { lots of these }

  abortcodes =
    { timelimit, userabort}; { The types of aborts that are processed
  abortset = set of abortcodes;

  var
    . { lots of these }

  aborted, timeout: boolean;
  abtcnt: integer;
  lastabort: real;

procedure rename(var f: textfile; lfn: scopelfn); extern;
{ This procedure changes scope file names by modifying their FETs.
  I really think this is the right way to specify the dynamic (run-time) association of a system file with a Pascal file.
  Overloading the reset and rewrite procedures and adding standards violating parameters to them seems so messy.
}

procedure interrupt(procedure inproc(reasons: abortset)); extern;
{ This procedure arms the SCOPE system routine 'reprieve' with a user supplied recovery routine. Time-outs and aborts are handled by this routine. Upon interrupt, the procedure passed as a parameter to the interrupt routine is invoked. After it executes, the program is restarted at the instruction where it was interrupted. By having the interrupt routine set global flags, controlled recovery is possible. }

{ about 140 additional procedures here.
  all written in quite Standard Pascal.

Note: Pascal-I has an interpreter that is similar to that of Pascal-S. In it, and in other procedures where the user might want to quit the actions of the program, loop terminators include a test of the aborted flag. Since Pascal-I has control of when aborts are acted upon, it does so only at convenient stopping places. For example, the interpreter only tests for aborts on user program statement boundaries. The state of Pascal-I and the interpreting user program always appear well defined. }
procedure timeoutsave;
{ This routine is called if a time out occurs. It is called by the main routine if the timeout flag is set during a recovery. Upon 'reprieve' invocation, enough additional time is allocated so that a user can save his/her program to a file. After exiting Pascal-I, more time can be requested (with ETL) or another login session started. The saved file allows the user to proceed from where he/she left off. }

var
  lfn: scopeLfn;

begin
  writeln(' You are out of time. Please enter the name of');
  writeln(' the file to which you want your program saved - ');
  if eos(input) then getseg(input); getch;
  { The eos (end of segment) and getseg (get segment) are rather unpleasant ways to interface to terminals. Fortunately, only a very few other places in Pascal-I have such code. Porting the program usually only requires defining null procedures for getseg and putseg and making eos return false. At one place, eos may need to be changed to eof. }
  getln(lfn); rename(textout, lfn); rewrite(textout);
  { get the file name and associate it with textout }
  saveblk(btabmax - 1, true); reset(textout);
  { write the program to it and rewind it for next time }
end { timeoutsave }

procedure intproc(reasons: abortset);
{ No Pascal procedure in Pascal-I calls this routine. It is invoked by the 'reprieve' service routine which is invoked by the system monitor when a time-out or user abort occurs. }

Incidentally, Pascal 6000 version 2 didn't have reentrant system routines. (The fault of using the RJ (return jump) to implement the calls.) Because this routine doesn't require any of the system routines to be accessed reentrantly, we can use a very simple version of the recovery routines in Pascal-I. Pascal-I is distributed with fully re-entrant recovery capabilities in its system routines.

const
  abtmintime = 2.0; { minimum time limit allowed between user recoverable aborts ( 2 secs.) }
  maxabtwocmd = 4; { maximum user aborts allowed between commands. If more then kill Pascal-I. }

var
  now: real;

function rtime: real;
  extern { real time clock Returns time in seconds, accurate to milliseconds. }
begin
  timeout := timelimit in reasons;
  aborted := userabort in reasons;
  if aborted then
    begin
      abtcnt := abtcnt + 1; now := rtime;
      if now - lastabort < abtmintime then
        begin
          message('* multiple aborts. ');
          goto 13 { bag it }
        end;
      lastabort := now;
    end;
  writeln; ich := "j
  { clear and restart I/O if abtcnt < maxabtwocmd then interupt(intproc); }
  if abtcnt < maxabtwocmd then interupt(intproc);
end { intproc }
The entire supplemental system routines are presented here. Bill Cheswick coded these for CDC's NOS operating system.

**rename** - change local file name.

```pascal
rename(ifet, name)
```

**interup** - set user-abort interrupt address.

```pascal
interup(proccaddr)
```

Of all the complex functions described, getting the real time took the most code to implement. Implementing Pascal-I on IBM, DEC and other systems proved easy because of the simplicity and isolation of the system dependent interface.
program LISP(input, output);

{ The essence of a LISP Interpreter.

Written by W. Taylor and L. Cox
First date started : 10/29/76
Last date modified : 12/10/76
}

label 1; { used to recover after an error by the user }
2 { in case the end the file is reached before a fin card }
const
maxnode = 600;
type

{ Inputsymbol =
( atom, period, (paren, rparen);
reservedwords =
(replac.hs, replac.tsym, headsym, talsym, eqsym, quotessym,
atom sym, consdys, headsem, lambdase m, copy sym, append sym, cons sym, consm sym);
statustype =
( unmarked, left, right, marked);
symbolicexpression = record

status: statustype;
next: symbolicexpression;
case an atom: boolean of
true: (name; alias);
case isreservedword: boolean of
true: (reservedword; case);
false: (head, tail: symbolicexpression);
}
end;

Symbolicexpression is the record structure used
to implement a LISP list. This record has a
field 'anatom' which tells which kind of node
a particular node represents (i.e. an atom or
a pair of pointers 'head' and 'tail').
'Anatom' is always checked before accessing
either the name field or the head and tail
fields of a node. Two pages ahead there are
two diagrams which should clarify the data
structure.

The global variables
var

{ Variables which pass information from the scanner to the read
routine }
lookaheadsym, { used to save a symbol when we back up }
sym: inputsymbol; { the symbol that was lost scanned }

{ id: alias; name of the atom that was last read }
freeptr: symbolicexpression;

{ the global lists of LISP nodes }
freelist, { pointer to the linear list of free nodes }
nodeLink, { pointer used to make a linear scan of all
the nodes during garbage collection }
alist: symbolicexpression;

{ two nodes which have constant values }

nilnode, node: symbolicexpression;

{ variables used to identify atoms with pre-defined meanings }
reservedwords: reservedword;
reserved: boolean;
reservedwords: array[reservedword] of alias;
freemoses: integer { number of currently free nodes known }
numberofedges: integer { number of garbage collections made }


begin
( the atom 'a' is
represented by --->

1

)

{ nilnode, tnode: symbolicexpression;
freellst, { pointer to the linear list of free nodes
statustype =
symbexpptr = "symbolicexpression;
resword: reservedwords;

symbolicexpression = record

status: statustype;
next: symbolicexpression;
case an atom: boolean of
true: (name; alias);
case isreservedword: boolean of
true: (reservedword; case);
false: (head, tail: symbolicexpression);
}
end;

procedure markeList;

{ for a particular node represents (i.e. an atom or
a pair of pointers 'head' and 'tail').
'Anatom' is always checked before accessing
either the name field or the head and tail
fields of a node. Two pages ahead there are
three diagrams which should clarify the data
structure.}

The alternative method, garbage collection, does not function
continuously, but is activated only when further storage is
required and none is available. The complete process consists of
two stages. A marking stage which identifies nodes still
reachable (in use) and a collection stage where all nodes in
the system are examined and those not in use are merged into
a list of available space. This is the technique we have chosen
to implement here for reasons of simplicity and to enhance the
interactive nature of our system.

The marking stage is theoretically simple, especially in LISP
programming systems where all records are essentially the same
size. All that is required is a traversal of the active list
structures. The most obvious marking system consists of a procedure
which makes a number of successive passes through the data
structure, each time marking nodes 1 level deeper into the tree
each pass. This is both crude and inefficient.

Another alternative procedure which could be used would use a
recursive walk of the tree structure to mark the nodes in use.
This requires the use of a stack to store back pointers to
branches not taken. This algorithm is efficient, but tends to
be self defeating in the following manner. The requisite stack could
become quite large (requiring significant amounts of storage).
However, the reason we are performing garbage collection in the
first place is due to an insufficiency of storage space. Therefore
an undesirable situation is likely to arise where the garbage
collector's stack cannot expand to perform the marking pass.
Even though there are significant amounts of free space waiting
for reclaiming.

A solution to this dilemma came when it was realized that space
in the nodes themselves (i.e. the left and right pointers) could
be used in lieu of the explicit stack. In this way the stack
information can be embedded into the list itself as it is traversed.
This algorithm has been discussed in Knuth and in Berzins: Data
Structures, Theory and Practice (2nd ed.), and is implemented below.

Since Pascal does not allow structures to be addressed both
with pointers and as indexed arrays, an additional field has been added
to sequentially link the nodes. This pointer field is set on initial
creation, and remains invariant throughout the run. Using this field,
we can simulate a linear pass through the nodes for the collection
phase. Of course, a marker field is also required.

procedure mark(list: symbolicexpression);

{ Variables which pass information from the scanner to the read
routine }

lookaheadsym, { used to save a symbol when we back up }
sym: inputsymbol; { the symbol that was lost scanned }

{ id: alias; name of the atom that was last read }
freeptr: symbolicexpression;

{ the global lists of LISP nodes }
freelist, { pointer to the linear list of free nodes }
nodeLink, { pointer used to make a linear scan of all
the nodes during garbage collection }
alist: symbolicexpression;

{ two nodes which have constant values }

nilnode, node: symbolicexpression;

{ variables used to identify atoms with pre-defined meanings }
reservedwords: reservedword;
reserved: boolean;
reservedwords: array[reservedword] of alias;
freemoses: integer { number of currently free nodes known }
numberofedges: integer { number of garbage collections made }


begin
( the atom 'a' is
represented by --->

1

)
Procedure nextsym reads the next symbol from the input file. A symbol is defined by
the global type 'symbol'. The global variable 'sym' returns the type of the next symbol read.
The global variable 'id' returns the name of an atom if the symbol is an atom. If the symbol is
a reserved word the global variable 'reserved' tells which reserved word was read.

begin
  if alreadypeeked
    then begin sym := lookaheadsym; alreadypeeked := false end
  else begin
    while ch = ' ' do begin
      if eoln(input) then writeln; read(ch); write(ch);
      end;
    if ch in ("\", ",", ")'' then begin
      case ch of
        "\": sym := (paren; name := id; isreservedword := reserved;
        ",": sym := period; i := 0; read(ch);
        ")": sym := atoL; i := 1; if i < 11 then id[ch] := ch; end
    end;
    if eoln(input) then writeln; read(ch); write(ch);
    until ch in ("", ",", ")'';
    reserved := replacechars;
    while id <> reserved(resword) do reserved := succ(resword);
    reserved := id = reserved(resword)
  end;
end;

procedure readexpr(var sptr: symbexpptr);

This procedure recursively reads in the next symbolic expression from the input file. When this procedure is called the global
variable 'sym' must be the first symbol in the symbolic expression to be read. A pointer to the symbolic expression read is returned
via the variable parameter sptr.

Expressions are read and stored in the appropriate structure
using the following grammar for symbolic expressions:

<s-exp> ::= <atom>.
      or (<s-exp> , <s-exp>) .
      (<s-exp> <s-exp> ... <s-exp> )

Where ... means an arbitrary number of, i.e. zero or more.

To parse using the third rule, the identity
(a . b . c ... z) = (a , b , c ... z)

is utilized. An extra left parenthesis is inserted into
the input stream as if it occurred after the imaginary dot.

When it comes time to read the imaginary matching
right parenthesis it is just not read (because it is not there).}

begin
  var
    nxt: symbexpptr;

  begin
    pop(sptra); nxt := sptr.next;
    case sym of
      <paren, period; error(1); atom
        with sptr do
          begin <atom>
            atom := true; name := id; isreservedword := reserved;
          if reserved then ressym := reserved
          else
            [paren
              with sptr do
                begin
                  ressym;
                  if sym = period then error(2)
                  if sym = <paren then sptr := nilnode ( () = nil )
                  else goto 1
                end;
          end;
      end;

  procedure readexpr;
The algorithm for this procedure was provided by Weisnam's LISP 5.1 Primer, p.125. This procedure prints the symbolic expression pointed to by the argument 'sptr' in the Lisp list notation. (The same notation in which expressions are read.)

```lisp
(procedure printname(name: alias))
  Procedure printname prints the name of an atom with one trailing blank.

  var
    i: integer;

  begin
    i := 1;
    repeat (write(name(i)); i := i + 1
    until (name(i) = ' ') or (i = 11);
    write(' ');
  end (printname);

(procedure printexpr(sptr: symexprptr))
  The algorithm for this procedure was provided by Weisnam's LISP 5.1 Primer, p.125. This procedure prints the symbolic expression pointed to by the argument 'sptr' in the Lisp list notation. (The same notation in which expressions are read.)

  label
    if
      (if sptr'.anatm then printname(sptr'.name)
      else write('"');
    1: with sptr do
      begin
        printexpr(head);
        if tail'.anatom and (tail'.name = 'NIL')
          then write('"');
          else write('[');
          if tail'.anatom
            then begin write('"'); printexpr(tail); write('"');
            else begin sptr = tail; goto 1 end
          end (printexpr);
    end (printexpr);

  end (if utility routines)

(The Expression Evaluator Eval)

(function eval(e, alist: symexprptr): symexprptr)
  Function eval evaluates the LISP expression 'e' using the association list 'alist'. This function uses the following several local functions to do so. The algorithm is a Pascal version of the classical LISP problem of writing the LISP eval routine for pure LISP. The LISP version of the code is as follows:

  (lambda (e alist)
    cond
      (atom e) (lookup e alist)
      (atom (car e))
        (cond ((eq (car e) (quote quote))
        (cdr (eval (cadr e) alist))
        (eq (car e) (quote car))
        (cons (eval (cadr e) alist)
        (eval (caddr e) alist))
        (eq (car e) (quote cons))
        (cons (eval (cadr e) alist)
        (eval (caddr e) alist))
        (eq (car e) (quote caddar))
        (cons (eval (cadr e) (car e))
        (eval (caddr e) (cdr e) (cons (caddr e) (caddar e)))))
      (eq (car e) (quote lambda))
        (eval (cons (caddr e) (cadr e) (cons (caddr e) (caddar e)))))
      (eq (car e) (quote label))
        (eval (cons caddar e (cadr e) alist))
      end (cond)
    end (eval)

The last three local functions, 'lookup', 'bindargs' and 'eval', are used by 'eval' to interpret a LISP expression.

(function replace(sptr1, sptr2: symexprptr): symexprptr)

begin
  if sptr1'.anatom then error(5) else sptr1'.head = sptr2;
  Replace := sptr1;
  end (replace);

(function replaceat(sptr, sptr2: symexprptr): symexprptr)

begin
  if sptr'.anatom then error(6) else sptr'.tail = sptr2;
  Replace := sptr1;
  end (replace);

(function head(sptr: symexprptr): symexprptr)

begin
  if sptr'.anatom then error(7) else head := sptr'.head
  end (head);

(function tail(sptr: symexprptr): symexprptr)

begin
  if sptr'.anatom then error(8) else tail := sptr'.tail
  end (tail);

(function cons(sptr1, sptr2: symexprptr): symexprptr)

var
  temp: symexprptr;

begin
  pop(temp); temp'.anatom := false; temp'.head = sptr1;
  temp'.tail = sptr2 cons := temp
  end (cons);

(function copy(sptr: symexprptr): symexprptr)

{ This function creates a copy of the structure pointed to by the parameter 'sptr'. }

var
  temp, nxt: symexprptr;

begin
  pop(temp); temp'.anatom := false; temp'.head = sptr;
  temp'.tail = nxt copy := temp
  end (copy);

(function append(sptr1, sptr2: symexprptr): symexprptr)

{ The recursive algorithm is from Weisnam, p.97. }

begin
  if sptr1'.anatom
    then
      if sptr1'.name <> 'NIL' then error(9)
      else append := sptr2
      else append := cons(copy(sptr1'.head), copy(sptr1'.tail), sptr2)
    end (append);
  end (append);

(function conc(sptr1: symexprptr): symexprptr)

{ This function serves as the basic concatenation mechanism for variable numbers of list expressions in the input stream. The concatenation is handled recursively, using the identity: conca(a,b,c1) = conca(a,conc(b,conc(c1))) }

The routine is called when a conc...command has been recognised on input, and its single argument is the first expression in the chain. It has the side effect of reading all following input up to the parenthesis closing the conc command. The procedure consists of the following steps:
  1. call with 1st expression as argument.
  2. Read the next expression.
  3. If the expression just read was not the last, recurse.
  4. Otherwise... unwind.

var
  sptr2, nilptr: symexprptr;

begin
  if sym = rparen
    then begin
      nextsym; readexpr(sptr2); nextsym;
      end (conc = cons(sptr1, conc(sptr2)));
  end (else);
  if sym = rparen
function atom(sptr:: symbexppt): symbexppt;
 begin
  var temp, nxt:: symbexppt;
  begin
    pop(temp); nxt := temp; temp := pop(alist);
    if sptr".name = sptr2".name then temp := n nod e;
    end
    if sptr".name = sptr2".name then temp := n nod e;
    end
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
    if sptr".name = sptr2".name then temp := n nod e;
    else temp := n nod e;
begin { LISP }
    writeln(' * EVAL *'); initialize; nextsym; readexpr(ptr);
    readln; writeln;
    while not ptr^.anatom or (ptr^.name <> 'FIN ') do
        begin
            writeln; writeln(' * value * '); printexpr(eval(ptr, alist));
            1: writeln; writeln; if eof(input) then error(11);
            ptr := nil;
        { call the } garbageman; writeln; writeln;
        writeln(' * EVAL * '); nextsym; readexpr(ptr); readln;
        writeln;
        end;
    writeln; writeln;
    writeln(' total number of garbage collections = ', numberofgcs: 1, ' .'
    ) ;
    writeln;
    writeln(' free nodes left upon exit = ', freenodes: 1, ' .');
    writeln;
end { LISP }.

Articles
An Implementation of New and Dispose using Boundary Tags

Branko J. Gerovac

The standard Pascal procedures New and Dispose are implemented using boundary-tag memory management. This implementation replaces the original New and Dispose modules in the run-time library of Oregon Minicomputer Software, Inc. Pascal-l which executes on Digital Equipment Corp. PDP-11 computers. Design details, although aimed at this configuration, should be generally useful. Performance of the original and boundary-tag implementations are analyzed and compared.

Key words: Pascal, New and Dispose, memory management, boundary tag.

1. Introduction

Many Pascal systems do not fully implement New and Dispose. One can speculate that (1) the full generality of New and Dispose was deemed unnecessary or undesirable, or that (2) efficient algorithms for New and Dispose are not readily available. This paper addresses the latter issue.

The standard Pascal run-time environment has two functionally different data storage areas: the stack and the heap.

The number of accessible data items on the stack is designated by the declarations of a program, and all operations that allocate and release stack storage and access stack data are implicit in program syntax. In addition, the block structure of a program designates the period (lifetime) during which stack storage is set aside.

In contrast, the number and lifetime of items on the heap are largely independent of program declarations, and heap operations are programmed explicitly. At run time, a program must (1) maintain access to heap data, by using pointers, and (2) allocate and release heap storage, by using New and Dispose.

Some Pascal systems implement the heap as a second stack (e.g., P-code Pascal [NAJN76]). A second stack requires that a program maintain the information necessary to release heap storage, and that heap storage is released in the reverse order from which it was allocated. This restriction may prevent the programmer from implementing algorithms that use a non-stack-like data structure [cf., HS76, HS78, W76].

Here, a boundary-tag scheme for managing free blocks permits an efficient implementation of New and Dispose. This module has many advantages over the original New and Dispose module in the run-time library of OMSI-Pascal-l [1]. OMSI-Pascal's original New and Dispose provided some insight into the problems of heap management. With the original module, examples of wide variation in memory efficiency and execution time are apparent. Since one of OMSI-Pascal's strong features is its applicability to real-time programming, many design decisions for the boundary-tag module were aimed at decreasing execution time. Memory efficiency improved also.

Performance analyses of each New and Dispose module are compared. Analyses of specific heap operations were carried out by calculating run times of each implementation. Simulation tests were run to obtain comparative performance during actual execution.

Although a specific hardware-software environment is discussed here, the design rationale would be appropriate for other systems. Pascal sources for each implementation of New and Dispose and assembly language sources for the boundary-tag module are provided to promote general use.

2. Description of the Original New and Dispose Module

The run-time memory configuration of OMSI-Pascal-l [ESI77], under DEC's RT-11 real-time operating system, is typical for block structured languages [NAJN76, AU77]. The operating system maintains areas of memory for interrupt vectors, system communication, the resident monitor and peripheral device registers [DEC78]. When a Pascal program is run, the program code is loaded into low memory, and then a Pascal run-time library routine initializes the data areas. The heap is located in low memory just above the program code and global storage, and the stack is located in high memory. The heap grows upward and the stack grows downward; the unused memory between the heap and the stack is available for expansion of either. No automatic memory-disk swapping of data occurs.

Two pointers are maintained by New and Dispose to manage heap memory: (1) $FREB$ points to the beginning of the unused area above the heap, and (2) $FRELL$ points to a list of free blocks within the heap. The free list is a singly linked list of blocks that have been disposed [2]. Each free block contains (1) a pointer to the next block in the list (a nil pointer if it is the last block in the list) and (2) the block's size. An advantage of the free list is that the information needed to manage a free block resides within the block itself. An additional memory overhead is required for free-block management. (Computers with virtual memory may benefit from a separate table of free blocks to avoid excessive memory-disk swapping.)

New. To allocate storage on the heap, program code passes the size needed to New [3]. (Appendix A contains Pascal sources of New and Dispose.) If one word is requested, it is allocated by extending the top of the heap by one word; one-word blocks do not fit on the free list because two words are necessary to contain pointer and size information. For a request of more than one word, the free list is searched for a block of the exact size (exact fit) of the block requested. If such a block is found, it is unlinked from the list and allocated; if no such block is found or the free list is empty, the heap is extended by the number of words needed to allocate the block. If collision with the stack results from extending the heap, program execution is terminated. The newly allocated block is zeroed to provide a clean slate and is used to help prevent inadvertent violation of the free list. New returns the address of the new block, and program code assigns this address to a pointer.

Dispose. To release storage to the heap, program code passes the address and the size of the block to Dispose. A block that is larger than one word is linked to the

Since New and Dispose may be called in any sequence, the heap can contain a mix of allocated and free blocks. The free list permits New to reuse free blocks. The size is always an even number of bytes due to the PDP-11's restriction that word-based data, e.g., integers, be stored at even byte (word) locations.
beginning of the free list and its size is recorded; a one-word block effectively is not released. Then, the free list is searched for a block adjacent to the top of the heap. If a block is found, it is released from the heap by unlinking it from the free list and decrementing $\text{KORE}$. This search is repeated until a full scan of the list is made without a decrease in the upper bound of the heap.

The original implementation of New and Dispose is uncomplicated, requires little code, and seems as though it would work well with typical Pascal programs. Generally, only a few different data sizes are specified in a program. The exact-fit allocation scheme often finds the size block needed in the free list; the size of the last disposed block is likely to be the same as the size of the next requested block, hence placement of the disposed block at the beginning of the free list may speed allocation. However, problems arise when worst-case memory space and execution-time performance are considered.

For example, since the free list does not keep track of disposed one-word blocks, one-word blocks limit the extent to which the upper bound of the heap can be reduced. Free blocks that are below a one-word block will never be adjacent to the top of the heap and cannot be released. Even so, Dispose continues to scan these free blocks. A simple solution would allocate two words for a one-word request so that the block would fit on the free list.

Another problem, easily fixed, is the unnecessary search that Dispose makes when a block is first linked to the free list. The free list need be searched only if the block currently being disposed is adjacent to the top of the heap.

Even with these changes, certain configurations of the free list generate inefficient memory use and a wide range of execution times.

Consider a program that places 100 blocks of one size in the free list. Suppose the program then requests a block of some different size. Since New employs an exact-fit algorithm, a search of the free list will not produce a block of the correct size and the heap will be extended for the new block. Effectively, 100 blocks of storage are not usable, the total size of the heap is larger than necessary, and the execution time of New has increased by the amount of time required to search 100 blocks.

Now consider that the 100 blocks were disposed in the reverse order from which they were allocated (last allocated, first freed). In other words, the blocks near the top of the heap are farther from the beginning of the list. When the final block (keystone) between the top of the heap and the 100 blocks on the free list is disposed, a chain reaction releases all 100 blocks from the heap. However, the full depth of the free list must be scanned for each block to be released. This results in a single call of Dispose that performs 5,050 comparisons, i.e., a complexity of $O(\sqrt{N}/2)$.

3. Selection and Design of a Heap Management Algorithm

In both cases described above, the large number of free blocks causes worse worst-case performance. This number can be reduced by merging adjacent free blocks. The resulting larger block would be available for allocation if its constituent blocks would have been too small. By allocating a portion of a large block and returning the remainder to the free list, the larger block is available for a variety of smaller size allocations. Thus, reusability of available memory is enhanced.

Since the heap grows toward the stack, the upper extent of the heap should be kept as low as possible. To accomplish this, blocks in the free list can be ordered by memory location; blocks which are nearer the bottom of the heap are placed closer to the beginning of the list. New, employing a first-fit search algorithm, allocates the lower free block of sufficient size. If the block exceeds the requested size, only the lower portion is allocated, and the remainder is returned to the free list. Biasing heap allocations toward lower memory helps avoid collision with the stack.

Dispose, then, maintains the ordered free list, and merges adjacent free blocks. Simply, when a block is disposed, a search is performed with blocks already in the free list to determine whether to merge the disposed block with a free block or to insert the disposed block into the free list; potentially, a full scan of the free list would be needed. However, literature on memory-allocation strategies [K73, S76, G76, H76, HS76] indicates that a dispose operation can be performed without scanning the free list by employing Knuth's "Boundary Tag" scheme for free-block management [K73]. The implementation presented here differs from Knuth's presentation in order to maintain the ordered free list.

The boundary-tag scheme uses two additional words of storage to mark the boundaries of each block; lower and upper boundary words are identical. Each boundary word contains the size of the block and a one-bit tag that signifies whether the block is allocated or free. Since the size is always an even number of bytes, bit zero can be used to tag the block. Bit zero is clear to indicate that the block is free and is set to indicate that the block is allocated. Dispose need check only the boundary.
words of the blocks adjacent to the block being disposed to determine whether a merge can be performed.

Each free block contains two pointers which enable access to the next and previous free blocks during insert and merge operations. Placement and referencing of the pointers was chosen to facilitate access using the auto-increment/auto-decrement addressing modes of the PDP-11 instruction set. Also, placement at the bottom of the block corresponds to fastest pointer referencing. (Although, placement of the pointers at the top of the block would seem advantageous when the lower portion is allocated, preliminary coding indicated a marked increase in code size and a very slight decrease in execution time.)

The heap is initialized with boundary blocks at the bottom and top of the heap. $\$FREE$ points to the lower boundary block, which is tagged as being allocated, and links the bottom and top of the free list into a circular list; the list can be traversed in either direction. If a large enough block is not found, then the heap has a size of zero. This is a pseudo block in that it is not linked into the free list; it serves only to provide a boundary word to check when the block adjacent to $\$FREE$ is being disposed. The boundary blocks eliminate the need for tests which otherwise would have to check boundary conditions during insertion on and removal from the free list. Without boundary blocks, Dispose would have required as many as 8 conditional tests to select from 12 separate operations. With the boundary blocks, only 4 tests and 6 operations are needed.

4. Description of the Boundary-Tag New and Dispose Module

The boundary-tag module was written so that no changes to the compiler or the rest of the run-time library would be needed (see Appendix Notes).

New. To allocate storage on the heap, program code passes the size of the block to New. (Appendix B contains Pascal sources of New and Dispose, and Appendix D, Macro-11 sources.) A request for one word is changed to two words. The free list is searched in either direction. If a large enough block is not found, then the heap is extended, providing that the heap does not collide with the stack. If a block which is larger than needed is found, the lower portion is allocated and the upper portion (remaining portion of the block) is returned to the free list. However, if the remainder would be too small to fit in the free list, the entire block is allocated. Then, the tags of the new block are set, the block is zeroed, and its address returned.

Dispose. To release storage to the heap, program code passes the address and the size of the block to Dispose; the size parameter is ignored since the actual size of the block is contained in the boundary word. The block's tag is checked to see that it is allocated and the block's address is checked to see that it is within the heap (OMSI-Pascal has been extended to permit pointers to data which are not stored on the heap). Then its tags are set to free, and the addresses of the lower- and upper-adjacent words are calculated. If the lower-adjacent block is free, the two blocks are merged; a merge with the lower-adjacent block has occurred. If neither adjacent block is free, the free list is scanned to compare the address of the block being disposed with the addresses of blocks in the free list. The disposed block is inserted in proper order, maintaining the ordered free list.

Problems in the original module have been corrected. One-word requests return a two-word block that will fit in the free list without special handling. Allocations are made from the lowest possible free block; the upper free blocks are more likely to be released from the heap. Free blocks are merged; the larger blocks are available for a variety of allocation sizes, and the shorter free list is more rapidly scanned. Boundary tags permit most blocks to be disposed without a scan through the free list.

5. Static Analysis

The additional operations of the boundary-tag module require more than twice the instruction space of the original. The number of storage words for each procedure is:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>New-org</th>
<th>New-tag</th>
<th>Dispose-org</th>
<th>Dispose-tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>38</td>
<td>103</td>
<td>33</td>
<td>78</td>
</tr>
<tr>
<td>Boundary</td>
<td>19</td>
<td>37</td>
<td>13</td>
<td>24</td>
</tr>
</tbody>
</table>

Execution-time equations for both New and Dispose modules were calculated using the instruction execution times given by the manufacturer for an LSI-11 with a 350 nanosecond microcycle time [DEC77]. Representative data, based on simulation tests (N=4, random) presented in the next section, are shown in brackets; all execution times are in microseconds (us). Subsequent references to the original implementation of New and Dispose and the Boundary-Tag implementation of New and Dispose are indicated respectively by New-org, Dispose-org, New-tag and Dispose-tag.

New-org performs three likely forms of allocation: (1) the free list is empty, allocate by extending the heap, (2) a free block of the correct size is found, allocate this block, and (3) the free list contains blocks that are not the correct size, allocate by extending the upper bound of the heap. The execution-time equations for New-org are:

1. free list empty 89.25 + 28.70L [433.65us]
2. allocate free block 76.30 + 30.80*Neworg + 28.70L [497.70us]
3. extend heap 117.95 + 30.80*Neworg + 28.70L [1232.35us]

Norg [25] the number of blocks on the free list.
Norg [2][5] the number of blocks searched to find one of the correct size.
L [12] the size in words of the newly allocated block, represents the size required to zero the block (the 28.70L term could be reduced to 11.90L).

The New-tag algorithm also performs three forms of allocation: (1) allocate an entire block from the free list, (2) allocate the lower portion of a block from the free list, and (3) allocate by extending the heap. New-tag:

1. entire free block 160.65 + 26.60*Newtag + 11.90L [303.45us]
2. portion of free block 207.90 + 26.60*Newtag + 11.90L [350.70us]
3. extend heap 176.05 + 26.60*Newtag + 11.90L [531.65us]

Ntag [8] the number of blocks on the free list.
Ktag [3] the number of blocks searched to find one of the correct size.
L [12] the size in words of the newly allocated block.

The advantage of New-tag results from the fewer blocks contained on its free list. In the 100 free-block example given in section 2, a single call of New-tag runs 3,542.35 us., while New-tag runs 378.00 us. The free list for New-tag contains only one block. Remember that New-org is extending the heap, while New-tag is reusing memory from the free list.

The Dispose-org algorithm has two major forms of releasing storage: (1) the block to the free list and do not decrease the upper bound of the heap, and (2) decrease the upper bound of the heap by the size of the block being disposed. Also, (3) worst-case execution time for a single call is the dispose of the keystone block described in section 2; representative time is given by Norg=25 for comparison with (1) and (2). Dispose-org:

1. add to free list 72.45 + 42.00*Norg [1,122.45us]
2. decrease heap 92.05 + 42.00*Norg [1,142.05us]
3. worst-case 72.45 + 42*(Sqrt(Norg)/2) + 61.60*Norg [14,737.45us]
The Dispose-tag algorithm has six forms of releasing storage: (1) scan the free list and insert the block without a merge, and (2) five forms of merging the block without a scan, the range and average of these are given. (3) The keystone dispose is not a worst case for Dispose-tag; it would execute as a merge operation. Instead, worst case is a full scan of the free list to insert the block at the bottom of the free list. Dispose-tag:

1. scan and insert  \[143.85 + 14.70(NTag/2) \text{ [202.65us]}\]
2. merge range \([134.05, 205.10]\) \text{[average 173.74us]}\]
3. worst-case  \[143.85 + 14.70NTag\] \text{[261.44us]}\]

An examination of the time needed to dispose an entire list shows the effect that multiple Dispose operations have on program execution. Assume a list of blocks is allocated and numbered in order of allocation \((1,2,3,X)\); the free list is initially empty. Two simple cases of disposing the list are: (1) LAFF—last allocated, first freed—blocks are disposed in the reverse order from which they were allocated \((X-3,2,1)\). Each call of Dispose decreases the upper bound of the heap. And, (2) FAFF—first allocated, first freed—blocks are disposed in the same order as allocation \((1,2,3,X)\). Each call of Dispose adds the block to the free list; the last call decreases the upper bound of the heap by the extent of the entire list. Also, worst case for each version of Dispose is: (3) LAFF-keystone, described in section 2 \((X-1), 3, 2, 1,X)\), is worst case for Dispose-org. And, (4) odd-LAFF/even-FAFF is worst case for Dispose-tag. The odd numbered blocks are disposed in reverse order, then all even numbered blocks are disposed in increasing order \((X-1), 5, 3, 1, 2, 4, 6, X)\); assume \(X\) is an even number. Each dispose of an odd numbered block must scan the entire free list to insert the block in order, the even numbered blocks merge with both lower- and upper-adjacent, and the \(X\)-th block decreases the upper bound of the heap by the extent of the list.

Dispose a list with \(X\) blocks \([X=100]\):

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>original</th>
<th>boundary tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LAFF</td>
<td>134.05 * X</td>
<td>[13, 405us]</td>
</tr>
</tbody>
</table>
| 2 | FAFF | \((134.05X)-(42)(5q(X)-X)/2)\)
  |    | \([355.60-(142.80X-2)/2])\) | [221, 305us] |
| 3 | LAFF-keystone | \((134.05X)-(42)(5q(X)-X)/2))\)
  |    | \([355.60-(142.80X-2)/2])\) | [431, 305us] |
| 4 | odd-LAFF/even-FAFF | \((134.05X)+(42)(5q(X)-X)/2)\)
  |    | \([355.60-(142.80X-2)/2])\) | [324, 205us] |

LAF and LAF-keystone are respectively the best- and worst-case examples for the original Dispose. The similarity of ordering between the two complicates the evaluation of run time for programs using the original module. While the original implementation of New and Dispose exhibits a wide range of execution times, the boundary-tag implementation is orderly even in the extreme examples.

6. Dynamic Analysis

Simulation tests were run to collect additional information on the comparative performance of the original and boundary-tag implementations of New and Dispose. The simulation program is similar to the one recommended by Knuth [K73] and is based on Monte Carlo techniques.

The test program runs in simulated time; the major loop of the program defines a simulated-clock tick. Briefly, at each clock tick: (1) All blocks that are at their lifetimes are disposed. (2) Then, a single block is allocated, its size and lifetime determined by generator functions. The allocated block is placed on a list that is ordered by lifetime limit. (3) Statistics on heap size and utilization and the numbers of allocated and free blocks are freed. (4) Then, a scan of the free list is executed. (5) The numbers of allocated and free blocks are recorded. Periodically, statistics and a frequency plot of memory use are output. The program continues until a simulated-time, a real-time, or a heap-size limit is reached; all tests reported here ran the full simulated-time limit of 25,000 ticks. At the end of the program, summary statistics and a frequency plot of memory use are output.

All tests were run with the same main program; only the generator functions for size and lifetime differed. A variety of generator functions were used. The functions were chosen so that the average allocated-block size was 12 words and so that the average number of allocated blocks was 50. A random number generator \(0.0, 0.99999\) serves as the basis for size and lifetime selection; the same sequence of random numbers was used for all tests.

Seventeen size functions were used. Each generated an even distribution of \(N\) block sizes \((N = 1...17)\) centered around 12 words. These 17 size functions are of the form:

\[\text{size}(N) = \text{Trunc}((\text{random}\times N + (12 - \text{Trunc}(N/2)))\]

The function for \(N=5\) requests allocations of 10, 11, 12, 13, or 14 words with equal probability. For \(N=4\), allocations of 10, 11, 12, or 13 are requested; functions for even values of \(N\) request blocks whose average size is 11.5 words.

Four lifetime functions were used: (1) Random, evenly distributed from 1 to 100 simulated-clock ticks, (2) Queue, fixed value of 50 ticks, (3) Stack, allocate 100 blocks, one per tick, then dispose all of them in the reverse order from which they were allocated, LAF, and (4) 80% Stack, lifetimes are 80% stack-like and 20% random. The equations for these functions are: (1) \(\text{size}(N) = \text{size}(1) \times (\text{random})\times N\) + (12 - \text{Trunc}(N/2))\)

Each size function \((17)\) was paired with each lifetime function \(4)\) to produce a test \((1\) of \(68)\) performed with each New and Dispose module. (Other tests produced similar results.) Statistics were gathered separately for each test-module combination.

Figure 1 plots the average number of blocks on the free list versus the size function for each test. Data points of the same lifetime function and New and Dispose module are connected. Each data point is the sum of the free-block counts from each simulated-clock tick averaged over 25,000 ticks. The free-block counts for the stack-lifetime tests were always zero and are not plotted.

Another way to view the results is to consider the ratio \((p)\) of free blocks to allocated blocks; the average number of allocated blocks is approximately 50 for all tests. In the random-lifetime curves, the boundary-tag module starts with \(p=5.4\%\) when \(N=1\) and increases to \(p=20.3\%\) when \(N=7\) where a plateau develops not rising above 24%; results with the original module begin with \(p=10.7\%\) when \(N=1\), \(p=72.6\%\) when \(N=7\) and continues to increase until \(p=20.3\%\) when \(N=7\). The other lifetime functions show an even greater difference between the two modules.

Figure 2 shows the average of total heap size divided by the number of allocated words, a measure of a module's memory-space efficiency. A value of 100 means that all words \((average 600)\) are allocated and that there is no additional overhead; the stack-lifetime tests with the original module show this performance. Even though there are no free blocks, stack-lifetime tests with the boundary-tag module show a 17% overhead due to the two boundary words needed for each block. Since the average allocated block is 12 words, 14 words actually are used; smaller or larger blocks
respectively raise or lower this overhead. The other lifetime tests show a correspondence between overhead and free blocks. The original module's overhead increases with increasing N while the boundary-tag module's overhead stabilizes.

Maximum heap size also closely corresponds to the number of free blocks and to the average heap size for the various tests. The maximum heap size for the original module was about 17% greater than average heap size, and the maximum for the boundary-tag module was 20% greater. However, maximum heap size for the original module was generally more than 20% greater than maximum heap size for the boundary-tag module.

Figure 3 presents the total run time of each test. Special hardware to measure only the run time of the New and Dispose operations was not available. The simulation program was revised to provide more meaningful run time; specifically, free blocks were not counted and statistics were not gathered since these measures vary between modules. The same random number sequence was used so that these statistical measures would be the same as in the previous tests with the unrevised program. The revised simulation program still included test-specific operations, such as calculation of lifetime and size of the block to be allocated and maintenance of the ordered-by-lifetime list of allocated blocks; however, since the test specific operations depend on the test performed rather than the New and Dispose module, a comparison between modules is meaningful even though comparisons between different test types may not be. Note that the run time difference between the original and boundary-tag modules on the same test is entirely due to the run times of New and Dispose.

The stack-lifetime tests contain the fewest test-specific operations and are considerably shorter than the other tests. The tests with other lifetime functions contain more test-specific operations and exhibit a shape similar to the previous two figures.

The boundary-tag module frequently maintains a smaller heap even though the two additional boundary words are needed per block. Thus, programs using the boundary-tag module are less likely to terminate from heap-stack collision. The boundary-tag module executes faster even though it involves more computation to allocate a portion of a larger block and to doubly link and order the free list.

The boundary-tag module's performance can be explained by the "systematic" memory-management strategy employed. The effects of the ordered free list, the first-fit allocation, and the allocation of the lower portion of a free block ensure the allocations are made as low as possible in memory; this results in a smaller heap and in maximal reuse of free memory. The boundary tags permit a merge of adjacent free blocks without a scan of the free list, and the resulting shorter free list permits a faster scan, when necessary. Similar results are analyzed more fully by Shore [577].

7. Future Directions

Fine Tuning

The boundary-tag New and Dispose module shows improved performance in execution time and free block count. However, the two boundary words per block sometimes can use a significant proportion of total memory. This is true only when the heap contains many small blocks. Can this overhead be reduced?

The current module optimizes execution time with the added boundary words; however, much of the boundary-tag module's improved performance can be attributed to merged adjacent free blocks, the ordered free list and first-fit allocation. It may be possible to modify or eliminate the boundary words with only a slight increase in execution time.

To permit separate tests of each modification, the module should be revised in stages that progressively simplify the structure of a heap block. First, remove the upper boundary word. Without this boundary tag, the dispose operation must always scan the free list. Second, remove the backward pointer and singly link the free list. Now, the free list can be scanned only forward. Currently, Dispose scans the free list from top to bottom in order to maintain the average depth of a scan; a block being disposed would seem to be nearer the top of the heap (a test of this supposition is
necessary, cf., [S77]). Finally, remove the lower boundary word. This lower boundary word contains the actual size of the block which may be slightly larger than the requested block. Remember that while a free block is being allocated if the upper portion is too large to fit on the free list, the entire block is allocated. Therefore, the elimination of the lower boundary word is not recommended.

Alternately, other methods of allocating small size blocks could be explored. Architectures which have large word sizes (32.64 bits) and limited byte addressing exhibit a greater memory-space overhead when small blocks are requested. One possible method (described using a 16-bit architecture) allocates a larger block, e.g., 16 words, and allocates successive requests of one word from this same block; an additional word in the block would "bit map" the allocated portions. When the block is full, another 16-word block would be allocated. This method would require a separate free list of these partially allocated blocks. This two-tier structure could be considered for 3, 5,... word blocks, also. Such an arrangement of heap structure could reduce memory-space overhead for small blocks while maintaining the advantages of boundary tags. Other improvements in the boundary-tag module may be possible in a different implementation environment.

Extensions

The boundary-tag module provides a fully general facility, permitting all typical uses of memory management. The heap becomes a perfect place to store objects whose size is run-time dependant.

The run-time system can make extensive use of the heap for I/O buffers, queues, etc. Small processor systems can use the heap for external code swapping instead of using the traditional overlay scheme. Demand paging (with random access files) can be implemented. For example, an Allocate procedure has been written with which a program can request any size block from the heap at run time. Allocate has been used to implement dynamic arrays accessed via a pointer.

Extensions to standard Pascal are used. The run-time system can make extensive use of the heap for I/O buffers, queues, etc. Small processor systems can use the heap for external code swapping instead of using the traditional overlay scheme. Demand paging (with random access files) can be implemented. For example, an Allocate procedure has been written with which a program can request any size block from the heap at run time. Allocate has been used to implement dynamic arrays accessed via a pointer.

The boundary-tag module provides the programmer with a powerful and efficient heap structure that not only implements standard Pascal effectively, but also permits applications that extend Pascal's scope.

Acknowledgment

I would like to thank William J. Melville and F. Garth Fletcher for their helpful comments on drafts of this paper.

References


Appendix

Notes

The Pascal code in Appendices A and B closely mirrors the actual run-time library sources which are in Macro-11 assembler code. The original New and Dispose Pascal sources are translated from OMSI-Pascal's run-time library.

Extensions to standard Pascal are used.

(1) Pointer arithmetic is used where necessary. A pointer is evaluated as a positive 16-bit integer, i.e., range 0.64K. Although addresses are actually in bytes, word addressing is generally used. The comment, '{ }', at the left margin marks pointer arithmetic.

(2) The construct, '"identifier', evaluates as the address of the storage location where the named object, 'identifier', is stored. Those familiar with OMSI-Pascal will recognize this extension. The comment, '"', at the left margin marks this usage.

In Appendix D, much of the documentation text has been removed. Most of the information has been covered in the body of this paper.

Persons wishing to install the boundary-tag module in their OMSI-Pascal should note that file open code (in S3 or SUPOM) uses storage on the heap without calling New. This code should be changed so that storage is allocated by an explicit call to New.
Appendix A—Original New and Dispose

```
definition

blockptr = "block;
block = record
next : blockptr; {--link to next free block--}
bsize : integer; {--size in words of block--}
filler : array [3..bsize] of word
end;

var
Free, Kore : blockptr; {--pointer to beginning of free list--}
Free = nil; Kore = blockptr; {--pointer to beginning of unused area--}

function New (size: integer) : blockptr;

var
scan, lastscan, scanA : blockptr;
begin{New

if (Free <> nil) and (size >= 2{words})

begin
lastscan := @Free; {--i.e., lastscan = Free--}
scan := Free;
while (scanA.next <> nil) and (scan <> nil) do

begin
lastscan := scan;
scan := scanA.next
end
end;

if (scan <> nil) and (size >= 2{words})

then

begin
Kore := scanA.next;
KoreA.next := scan; {--link to beginning--}
KoreA.prev := @Free;
end

scan := scanA.next;
Kore := Kore + size;
begin {--extend heap for new block--}
scan := Kore;
Free := scanA.next;
end {New

end {New

function Dispose (P : blockptr; size: integer);

var
scan : blockptr;
begin {Dispose

if (P <> nil) and (size >= 2{words})

then

begin
scan := P; {--set up free block--}
scanA.next := Free;
Free := scan;
end {Dispose

end {New
```

Appendix B—Boundary-Tag New and Dispose

```
definition

type
blockptr = "block;
block = record
next : blockptr; {--link to next free block--}
bsize : integer; {--size in words of block--}
filler : array [3..bsize] of word
end;

var
Free, Kore : blockptr; {--pointer to boundary block at bottom of heap--}
Free = nil; Kore = blockptr; {--pointer to boundary block at top of heap--}

function New (size: integer) : blockptr;

var
scan, lastscan, scanA : blockptr;
begin {New

if (Free <> nil) and (size >= 2{words})

then

begin
lastscan := @Free; {--i.e., lastscan = Free--}
scan := Free;
while (scanA.next <> nil) and (scan <> nil) do

begin
lastscan := scan;
scan := scanA.next
end
end;

if (scan <> nil) and (size >= 2{words})

then

begin
Kore := scanA.next;
KoreA.next := scan; {--link to beginning--}
KoreA.prev := @Free;
end

scan := scanA.next;
Kore := Kore + size;
begin {--extend heap for new block--}
scan := Kore;
Free := scanA.next;
end {New

end {New

function Dispose (P : blockptr; size: integer);

var
scan : blockptr;
begin {Dispose

if (P <> nil) and (size >= 2{words})

then

begin
scan := P; {--set up free block--}
scanA.next := Free;
Free := scan;
en
```
if (Free = nil) then begin --this is the first New call--
  Initialize_heap;
else begin --search free list for first-fit--
  repeat
    scan := scanA.next;
  until ( (scan = Free) or (scanA.lsize >= size) );
if (scan = Free) then begin --did not find a large enough free block--
  scan := Kore + 1(words);
  Kore := Kore + size + 2(words);
  if ( (Stack <= Kore) and (Stack > Free) ) then
    begin
      fatal_error("Out of Memory");
      KoreA.lsize := 0, KoreA.ltag := freed;
    end
  else if ( (scanA.lsize >= (size + 2(words)) ) ) then begin
    --found a free block that is too large--
    scan := scanA.next;
    remscanA.lsize := scanA.lsize;
    remscanA.ltag := freed;
    remscanA.usize := remscanA.usize;
    remscanA.utag := freed;
    remscanA.next := scanA.next;
    remscanA.prev := scanA.prev;
    remscanA.nextA.prev := remscan;
    remscanA.prevA.next := remscan
  end
else begin
  --found a free block just about the right size--
  scan := scanA.next;
  scanA.nextA.prev := scanA.prev;
  scanA.prevA.next := scanA.next;
end;

New := scan;
scanA.lsize := size, scanA.ltag := alloc;
scanA.usize := size, scanA.utag := alloc;
--clear the new block--
for i:=size downto 1 do scanA.filler[i] := 0
end;

procedure Dispose (P : blockptr);
{--50 not need size parameter because--}
{--boundary words contain actual size--}
var
LA, UA, scan : blockptr;
begin
if ( (P < Free) or (P > Kore) ) then warning('not a heap pointer')
else if ( (P < nil) and (PA.ltag < freed) ) then
begin
  LA := P + 2(words) - LAA.usize;
  LAA.usize := LAA.usize + 2(words);
  LA := LA
end
end;
Error handling receives only brief mention since its implementation depends on the facilities of the total Pascal system; however, a few problems with memory management and pointers, in general, are worth consideration (cf., [FL77]).

Correct operation depends on the integrity of the information stored to manage memory; a program that writes outside of an allocated block can corrupt management information. To prevent corruption, bounds checking should be incorporated in the Pascal implementation (bounds checking is available in OMSI-Pascal V1.1). However, a few additional tests in the boundary-tag module may provide information on the cause of a failure and possibly show how to continue program execution.

During Dispose, a block’s upper and lower boundary words can be compared; a difference indicates an out-of-bounds access. The size parameter, which approximates the actual block size, can be used to examine adjacent blocks and a difference indicates an out-of-bounds access. The size parameter, which approximates the actual block size, can be used to examine adjacent blocks and possibly to reconstruct the boundary words. In addition, since the free list is ordered, the pointers can be checked for proper order. With a short free list, these tests would not incur a great time overhead. If the free-list links have been overwritten, the entire heap could be scanned by use of the size field in the boundary words. Sometimes regeneration of the free-list links and correction of mismatched boundary words may be possible; in most cases though, little can be done, except to terminate program execution.

Dangling pointer references also pose a problem. Compiler generated code passes the address of the block to be disposed and leaves the pointer to this block unchanged. In other words, the pointer points to a free block giving the program direct access to the free list. Dispose should be able to reference the pointer so that its value can be set to nil. When there are multiple pointers to the same block, however, the other pointers continue to reference the free list, even though the disposed pointer may be set to nil. A solution requires redesign of pointer implementation.
\begin{verbatim}

mov r0, -(sp)
mov r1, r3
cmp r0, #3
bhi 79

mov r1, r0
mov r0, -(sp)
add r1, r0
dec r1
mov r0, (r1)
ccc r0, r1

for i:-size in words down to 1 do
scanA.prevA.next:aremscanj
remscanA.nextA.prev:-remscanj
size:-scanA~size;
err1.1<>O
{N
allocate_from_$KORE
err1.1<>O
then
{N
allocate_from_$FREE
endfor;

mov (sp), r3
mov (sp), r2
mov (sp), r0
rts pc

end;

err0:
.asciz "?Fastlib-P-NEW-Out of Memory"/
.even
.endc
.end;

dsabl lsb

 sbtll $872 : Dispose with boundary tag

distag version: 1.1 : 16-Mar-79 ; check for pointer not to heap
distag version: 1.0 : 03-Oct-78

Calling sequence:

; dispose(f);
mov p(r0), -(sp)
jsr dispose
mov p(r0), -(sp)

end;

even
.endc
.end;

\end{verbatim}
proc DISPOSE(P:pointer);
begin

   if P=ntl then goto return endif;
   if (warn_d1<>O) then
      if P<$FREE or P>$KORE then warn(not a heap ptr) endif;
   endif
   if PA.ltag-free then goto return endif;
   pA.ltag:=free;
   if LA.utag-free then warn(not a heap ptr) endif;
   LA.utag:=free;
   if LA=lower_adjacent(P) then
      merge(LA,P)
      if POLA then
         PA.prev:=LA.prev;
         pA.next:=P
      endif;
      LA.next:=UAA.next;
      LA.prev:=scan
      LA=scanA.prev
      repeat
         scan:=scanA.next;
         scanA.next:=P;
         PA.nextA.prev:=PA;
      until:
      PA.next:=scan;
      PA.prev:=scanA.prev;
      PA.nextA.prev:=PA;
      endi;
   endif;
   return
end;

.....
Dear Mr. Cichelli,

We are of course happy to submit the QPP article for publication in Pascal News. (Actually, being a member of PUG myself, I should have thought of sending you the article earlier.)

Enclosed is a copy of the SIGPLAN article together with the code implementing the external procedures on the Nord.

Sincerely,

Terje Noodt
2. It must be possible for processes to exchange information - that is, one process must be able to access the attributes of another process.

To transform the procedure concept into a process, point 1. requires that the attributes of a "process-procedure" must be retained while it has a passive phase. That is, a "process-procedure" cannot execute on the stack top as usual, but must have some permanent space in memory.

Point 2. requires some form of looking "into" a procedure. In Pascal, a similar mechanism is given by the record concept. Consider the following program fragment:

```pascal
type
  PROCESS = record
    x, y: T
  end;
  PTRPROCESS = PROCESS;
var
  p: PTRPROCESS;

procedure processprogram;
var
  LOCALS: PROCESS;
begin
  with LOCALS do
  begin
    . . .
  end
end
```

Within the `with` statement in `processprogram` the attributes `x` and `y` may be accessed directly.

A process is created by calling the function

```pascal
function NEWPROCESS(procedure PROG);
```

This function allocates data space for the procedure PROG on the heap. The function value is a pointer to the record containing the process attributes. In reality, the pointer is a reference to the inside of the procedure object. The Pascal system, however, treats the pointer as if it were generated by the `NEW` function.

The main program (or another process) may access the attributes through the pointer generated by `NEWPROCESS`.

The following program fragment shows how a process is generated, and its attributes accessed from the outside:

```pascal
  p := NEWPROCESS(processprogram);
  . . .
```

Several processes of the same type may be generated as follows:

```pascal
var
  p1, p2: PTRPROCESS;
  p1 := NEWPROCESS(processprogram);
  p2 := NEWPROCESS(processprogram);
```

Processes of different types may be defined by declaring different `PROCESS` types, or by defining a variant part for each type of process within `PROCESS`.

Thus, a usable process concept has been established by

1. Implementation of the function `NEWPROCESS`. In Nord-10 Pascal this is an assembly routine of 15 instructions.
2. Requiring that the programmer stick to the following rules:
   a. Define a record type `PROCESS` which contains those variables of a process which are to be visible from outside the process.
   b. Declare a variable `LOCALS` of type `PROCESS` as the first variable within the process procedure.
   c. Surround the statements of the procedure by `with LOCALS do begin . . . end`

2.2 Sequencing

It must be possible to start and stop the execution of any process, in order that operations occur in the sequence required by the actual application. For this purpose, two operations are implemented (these are modelled after the corresponding primitives in Simula 67):

```pascal
procedure RESUME(p: PTRPROCESS);
```

This procedure transfers control from the caller to the process given by the actual parameter `p`. The execution of `p` is resumed at the place where the process last became passive. The caller becomes passive.

```pascal
procedure DETACH;
```
When a process p calls DETACH, it becomes passive. Control goes to the last process x which called RESUME(p).

The following method has been used to implement RESUME and DETACH efficiently and with ease.

A Pascal procedure object will normally contain one location for the return address (RA), and one location for the dynamic link (DL). Let CP be a pointer to the currently active process, and consider the main program to be a process with the name MAIN.

The operation RESUME(p) leaves the current program address in CP.RA, and the address of the currently active object (which may be CP itself or an ordinary procedure called by CP) in CP.DL. p.DL becomes the new active object, and execution is resumed at p.RA.

The DETACH operation is restricted to be used to give control back to the main program. It leaves the current program address in CP.RA, and the address of the currently active object in CP.DL. MAIN.DL becomes the new active object, and execution is resumed at MAIN.RA.

The DL location of a process is zero while the process is executing. Thus, CP is found by following the DL chain until DL equals zero. The following function is provided to enable the Pascal program to find CP:

```
function THISPROCESS: PTRPROCESS;
```

### 2.3 Summary

With a very small effort a primitive but usable process concept has been implemented within Pascal. On the Nord-10, the routines NEWPROCESS, RESUME, DETACH and THISPROCESS consist of ca 60 assembly instructions. No changes have been made to the Pascal compiler or the Pascal run-time library. Although Pascal may operate differently on other computers, the authors believe that our method of implementation may be adapted to most Pascal systems.

On the Nord-10, an ordinary procedure called from a process will execute in the memory space allocated to that process. This requires that the process object be large enough to accommodate such procedure calls. We have solved this problem by letting NEWPROCESS have one extra parameter, giving the largest necessary space for the process.

### 3 Process Scheduling

Section 2 defines and indicates how to implement a process concept and the basic primitives for process sequencing. To program a real-time system or a simulation model, some additional concepts are needed. Also in this case SIMULA 67 is used as a source of inspiration. The new programming platform contains:

- a system time concept.
- a "sequencing set" containing the processes scheduled for future execution.
- primitives for process scheduling.

In this section we show how these concepts may be implemented in Standard Pascal, using the basic primitives of section 2.

#### 3.1 Simulated time, Real time

In the case of simulations, the system time is introduced as in SIMULA; but in a real-time environment the system time corresponds closely to the time defined by the computer's real-time clock. The system time is represented by a variable in the main program:

```
SYSTIME:real;
```

The execution of an active phase of a process, called an event, is regarded as not consuming system time. That is, SYSTIME is only updated between the events. How SYSTIME is updated is described below.

#### 3.2 The sequencing set

A process may be scheduled for the execution of a future event. An event is associated with a system time, indicating when the event will occur. This time is represented by a variable local to each process:

```
EVTIME:real;
```

All scheduled processes are collected in a set, the sequencing set, sorted on the EVTILINE variable. The sequencing set is represented by a main program variable:

```
SQS:PTRPROCESS;
```

which points to the first member of the set, and a variable

```
NEXTPR:PTRPROCESS;
```

in each process pointing to the next element of the sequencing set.

When an active phase of a process ends, the first process P in the SQS will be the next process to execute an event. The value of SYSTIME is changed to EVTIME of P. If simulated time is used, the simulation is carried on by resuming the process P.

In a real-time system the new value of SYSTIME is compared with the computer's clock. If the difference is positive, the Pascal program makes a monitor call to release the use of the
CPU for the given amount of time. On return from the monitor call the procedure RESUME(P) is called.

3.3 Process scheduling

The following procedures define a small but convenient set of operations for discrete event scheduling. All procedures are written in Standard Pascal. The amount of Pascal code is about 40 lines. For a detailed description see the appendix.

procedure PASSIVATE;

The caller process ends its active phase, and the next event is given by the first element of the SQS. SYSTIME is updated, and in the real-time case the program may request a pause before the next process is resumed.

procedure HOLD(del:real);

Equivalent to PASSIVATE, except that the caller is put into the SQS with an event time equal to SYSTIME+del.

procedure ACTIVATE(p:PTRPROCESS; del:real);

The process p is scheduled to have an event at the time SYSTIME+del.

procedure CANCEL(p:PTRPROCESS);

If the process p is scheduled to have an event, this event is cancelled. That is, p is removed from the SQS.

3.4 Summary

Based on the basic primitives discussed in section 2, we have defined a set of additional primitives suitable for discrete event scheduling. These primitives are implemented by Standard Pascal procedures and data structures. The system time concept is introduced in two variations: simulated time and real time. In the implementation the difference between the two time concepts is only visible as a small modification of the procedure PASSIVATE. An important consequence is that it is possible to test out a program by simulation and afterwards use the same program as a part of a real time system.

4 Concluding remarks

As an example, the Bounded Buffer problem has been programmed in the appendix.

At the University of Oslo, QPP has been used to program the UNINETT node. UNINETT is a computer network of the central computers of all universities in Norway, plus several other governmental computers. Each institution has a node machine which hooks one or more computers into the network. At the University of Oslo, this node is a Nord-18. The size of the UNINETT node program is about 2280 lines of QPP code. In the development of this program, keeping to the restrictions of QPP was neither hampering nor the cause for any serious problems. The UNINETT project has shown that a considerable amount of development time may be gained by going from assembly code to a "primitive" high-level language tool. In cases where a full-fledged language tailored to the actual application (such as Concurrent Pascal) is not available, there seems to be good reason to select a solution such as ours.

The UNINETT node program was developed on a Nord-18 running the MOSS operating system. The first step in testing the program was to run it under MOSS as a simulation, using simulated time. Then the program was run in real time under MOSS. Finally, the program was transported to the UNINETT node machine, where it runs in real time. The node machine has a rudimentary operating system only, which supports stand-alone systems of this kind. The small size of the code which implements the QPP process primitives, has allowed us to easily make different versions to adapt to the environment in which the UNINETT program was to be run. It has proved very valuable to run the program as a simulation before it was run in real time. Development time was also saved by testing under an operating system with utilities such as interactive debugging, a file system etc. The errors remaining after transporting the program to the node machine have been few.

The reader who compares QPP with for instance Concurrent Pascal, will remark that QPP contains no primitives for the protection of shared data. Such a mechanism could be useful in QPP, but is not strictly necessary. The reason is that processes run in quasi-parallel rather than true parallel. An active phase of a process is regarded to take zero time, and thus is an indivisible operation. Time increases only when control is transferred from one process to another. It is the programmer who decides at which points in the program this may occur.

Appendix

This appendix contains a simple example of the use of QPP. A producer process generates characters which are read by a consumer process. The rate of production is up to the processes themselves, and in order to remove some of the time dependency between the processes, they are connected by a bounded buffer. However, since the buffer may get full (or empty) there is still need for some synchronization of the processes. This is achieved by the use of the ACTIVATE and PASSIVATE primitives.
The program also contains a complete implementation of the concepts defined in section 3. Names corresponding to concepts and primitives in QPP are written in capital letters, while small letters are used for variables particular for the example.

```pascal
program prodcon;
const
  buflen = 16;
  buflgml = 15;

(* definition of bounded ring buffer *)
bufindex = 0..buflgml;
buf=record
  p,c:bufindex;
  txt:packed array[bufindex] of char;
end;
ptrbuf=1buf;

(* definition of the data structure of the processes *)
PTRPROCESS=1PROCESS;
process type=(producer,consumer);

PROCESS=record
  var
    NEXTPR:PTRPROCESS;
    EVTIME:real;
    INSQS:boolean;
  case processtype of
    producer: (outbuf:ptrbuf; outcha:char);
    consumer: (inbuf:ptrbuf; incha:char);
  end;
end;

var
  SQS:PTRPROCESS;
  SYSTIME:real;
  ptrpro,ptrcon:PTRPROCESS;

(** basic primitives **) 

function NEWP(procedure p; siz:integer):PTRPROCESS; extern;
function THISP:PTRPROCESS; extern;
procedure RESUME(p:PTRPROCESS); extern;
procedure DETACH; extern;

procedure INTOQS(p:PTRPROCESS);
var rp, rp0:PTRPROCESS;
begin
  with p do
  begin
    rp:=SQS; rp0:=nil;
    while (rp<>nil) and (rp.EVTIME<EVTIME) do
      begin rp:=rp; rp:=rp.NEXTPR end;
    if rp0=nil then SQS:=p else rp0.NEXTPR:=p;
    NEXTPR:=rp; INSQS:=true
  end;
end;

procedure CANCEL(p:PTRPROCESS);
var rp, rp0:PTRPROCESS;
begin
  with pt do
  begin
    rp:=SQS; rp0:=nil;
    while rp<>p do begin rp:=rp; rp:=rp.NEXTPR end;
    if rp0=nil then SQS:=rp.NEXTPR else rp0.NEXTPR:=rp.NEXTPR;
  end;
end;

procedure PASSIVATE;
var p:PTRPROCESS;
begin
  p:=SQS; if p=nil then DETACH else SYSTIME:=p.EVTIME;
  (* if realtime then monitor call PAUSE(SYSTIME-CLOCK) *)
  SQS:=p.NEXTPR; pt.INSQS:=false; RESUME(p)
end;

procedure HOLD(del:real);
var p:PTRPROCESS;
begin
  p:=THISP; pt.EVTIME:=SYSTIME+del; INTOQS(p); PASSIVATE end;

procedure ACTIVATE(p:PTRPROCESS; del:real);
begin
  CANCEL(p); pt.EVTIME:=SYSTIME+del; INTOQS(p) end;
```
(** buffer routines **)  

function bufempty(bp:ptrbuf:boolean;  
begin bufempty:=(bpt.p=bpt.c) end;  
function buffull(bp:ptrbuf:boolean;  
begin with bp do  
if ((p+1) mod buflength)=c then putchar:=false else  
begin txt[p]:=ch; p:=(p+1) mod buflength; putchar:=true end;  
function putchar(bp:ptrbuf:ch:char)boolean;  
begin with bpt do  
if (p+l) mod buflength)=c then putchar:=false else  
begin txt[p]:=ch; p:=(p+1) mod buflength; putchar:=true end;  
function getchar(bp:ptrbuf:var ch:char)boolean;  
begin with bpt do  
if p=c then getchar:=false else  
begin ch:=txt[c]; c:=(c+l) mod buflength; getchar:=true end;  
(** processes **)  
procedure pproducer;  
var LOCALS:PROCESS;  
begin DETACH;  
with LOCALS do  
while true do  
begin (* produce next character *)  
if bufempty(outbuf) thenactivate(ptrcon,0);  
while notputchar(outbuf,outcha) dopassivate  
end;  
procedure pconsumer;  
var LOCALS:PROCESS;  
begin DETACH;  
with LOCALS do  
while true do  
begin (* consume character *)  
if buffull(inbuf) then activate(ptrpro,0);  
while not getchar(inbuf,incha) dopassivate;  
end;  
(** main program **)  
begin  
ptrpro:=NEWP(pproducer,100); ptrcon:=NEWP(pconsumer,100);  
new(ptrpro.outbuf); ptrcon.inbuf:=ptrpro.outbuf;  
RESUME(ptrpro);  
end.

Q P P  
RUN-TIME ROUTINES TO TRICK THE NORD PASCAL SYSTEM INTO TREATING QUASI-PARALLEL PROCESSES  
(IN THIS VERSION THE RESTRICTION THAT DETACH MAY RELINQUISH CONTROL TO THE MAIN PROGRAM ONLY, HAS BEEN REMOVED)  
PROGRAMMER: T. NOODT, COMPUTING CENTER, UNIV. OF OSLO  
DATE: JUNE, 1980  

NOTE:  
1. THE NORD-10/180 REGISTERS ARE:  
P PROGRAM COUNTER  
L LINK REGISTER  
X POST-INDEX REGISTER  
B PRE-INDEX REGISTER  
T TEMPORARY REGISTER  
A ACCUMULATOR  
D EXTENDED ACCUMULATOR  
2. THE B REGISTER CONTAINS A POINTER TO THE CURRENTLY ACTIVE OBJECT + 200 OCTAL.  
3. WHEN A ROUTINE IS CALLED, THE PARAMETERS ARE FOUND AT ADDRESS (B) + (A) + N, WHERE N=4 FOR FUNCTIONS, N=3 FOR PROCEDURES.  
4. A FUNCTION RESULT IS TRANSFERRED IN A.  

NEWP= SWAP SA DB  
RADD SA DB  
STA SAVB,B  
COPY SL DA
STA  SAVL,B  % SAVE POINT OF CALL
COPY  SB DX
LDA  PARAM+1,B  % GET SIZE
AAA  2  % ADD SPACE FOR RETB AND RETP
JPL  I  (5PNEW  % CALL NEW TO GET OBJECT
LOX  0,B  % OBJECT POINTER
AAX  2  % ADJUST POINTER PAST RETB AND RETP
LDA  PARAM+1,B  % P'S STATIC LINK
STA  STLK,X
LDA  SAVL,B
STA  RETP,X
LDA  SAVB,B
STA  RETB,X
STZ  DYLK,X  % INDICATE ACTIVE PROCESS
LDT  PARAM+2,B  % P'S CODE
AAT  4  % SKIP FIRST 4 INSTRUCTIONS OF P
% (THEY DO NON-RELEVANT CHECKS)
COPY  SX DA
AAA  3  % "RECORD" POINTER
% (RETERS TO FIRST LOCAL VARIABLE)
COPY  SA DB
AAB  175  % STACK POINTER
COPY  ST DP  % EXECUTE PROCESS
% (GENERATE LITERALS)
)FILL
5PESH= *  % IGNORE THE USUAL STACK-HEAP OVERFLOW CHECK
EXIT  %

)9END

)9BEG  THISP  %
9ENT  THISP  %
% FUNCTION THISP: PTRPROCESS;
% THISP= *
COPY  SB DX  % FOLLOW DYNAMIC LINK
LDA  DYLK-200,X  % UNTIL IT IS ZERO (=PROCESS FOUND)
JAZ  *+3
COPY  SA DX
JMP  *-3
COPY  SX DA
AAX  -175  % ADJUST POINTER BY -200+3
EXIT  %

)9END

)9BEG  RESUME  %
9ENT  RESUME  %
% PROCEDURE RESUME(PTR: PTRPROCESS);
% RESUME= *
COPY  SB DX  % PTR
LDX  3,X,B  % TOP OF OBJECT
AAX  -3
COPY  SL DA
STA  RETP,X  % RETURN POINT

COPY  SB DA  % RETURN OBJECT
STA  RETB,X  % ACTIVE OBJECT INSIDE PROCESS
LDA  DYLK,X  % INDICATE ACTIVE PROCESS
COPY  SA DB
STA  DYLK,X
LDA  LSC,X
COPY  SA DP  % JUMP

)9END

)9BEG  DETACH  %
9ENT  DETACH  %
% FUNCTION DETACH: PTRPROCESS;
% DETACH= *
COPY  SB DX
LDA  DYLK-200,X
JAZ  *+3
COPY  SA DX
JMP  *-3
AAX  -200
COPY  SB DA
STA  DYLK,X
COPY  SL DA
STA  LSC,X
LDA  RETB,X
COPY  SA DB
LDT  RETP,X
COPY  SX DA
AAA  3
COPY  ST DP  % PROCESS PTR (FUNCTION RESULT)
)9END

)9BEG  DISPP  %
9ENT  DISPP  %
9EXT  5PDSPE  % PROCEDURE DISPP(VAR PTR: PTRPROCESS);
% DISPOSE PROCESS
% MAY BE INCLUDED IF DYNAMIC DEALLOCATION OF PROCESSES IS
% WANTED, AND THE PASCAL SYSTEM HAS THE DISPOSE PRIMITIVE.
% DISPEP= *
COPY  SA DX
LDX  3,X,B  % GET pointer to PTR
LDA  0,X
STA  0,X  % PTR := NIL
AAX  -5
SAX  177
RADD  SB DX
STA  0,X  % ADJUST TO TOP OF ALLOCATED OBJECT
JMP  I  (5PDSPE  % TRANSFER PARAMETER TO DISPOSE
% CALL DISPOSE
)FILL

)9END

)9EOF
Hi,

I understand that the Pascal Users Group is interested in putting together a package of software tools. We of the Software Tools Users Group are doing much the same thing. We have some 50-60 tools (editing, text manipulation, formatting, sorting, command line interpreter, etc.) which simulate the Unix environment and originated from the little book Software Tools by Brian Kernighan and P. J. Plauger. The tools are currently written in ratfor, a portable Fortran-preprocessor language, and running on everything from an 8080 to a Cray. Our users group has a mailing list of almost 700 and holds meetings twice a year.

There have been several people in the group interested in translating the tools into Pascal. One man has already hand-coded a few of them in Pascal. Another group in England has used a mechanical translator written in Snobol to transfer the tools into KCEPL. I think a similar translator could be developed to translate into Pascal. If people in your group were interested in our tools, perhaps we could work together to build such a translator.

I've enclosed an LBL Programmers Manual to give you an idea of what we have available. Other sites also have nice tools—University of Arizona and Georgia Tech. have good packages too. I've also sent along our newsletters to give you an idea of what the users group is doing.

Even if translation of our tools into Pascal doesn't seem feasible, do let me know if you think there might be other ways our groups could work together.

Sincerely,

Debbie Scherrer
Co-ordinator, Software Tools Users Group
Dear Mr. Shaw:

I maintain PASCAL 6000 Version 2 and Version 3 at NASA, Langley Research Center, Hampton, Virginia. I have made several modifications to our compilers to enhance the usability of the compilers without changing the language itself. I am writing to describe briefly one such modification because it is easily implemented and may be useful to other installations.

This modification introduces a new option to the compiler which displays the locations of the fields within a record when invoked. Following each record type declaration, the field identifiers with their relative locations in the record are given. The following is an example of the output generated by our compiler with the option invoked:

```
3 REC = PACKED RECORD
4  FIELD1: CHAR
5  FIELD2: CHAR
6  FIELD3: INTEGER
7  FIELD4: PACKED ARRAY[1..200] OF BOOLEAN
8 END;

FIELD01: 0<59,54>  FIELD02: 0<5,0>
FIELD03: 1<59,0>  FIELD04: 2<59,> - 5<40>

9 VAR:
10  VREC: RECORD
11    STORAGE1: INTEGER
12    STORAGE2: CHAR
13    STORAGE3: BOOLEAN
14    STORAGE4: REAL
15 END:
16  STORAGE1: 0<59,0>
17  STORAGE2: 1<5,0>
18  STORAGE3: 2<0,0>
19  STORAGE4: 3<59,0>
```

The formats used above have the following meanings:

- \( W<B1,B2> \) indicates the field is in word \( W \) relative to the start of the record and uses bits \( B1 \) through \( B2 \).
- \( W1:B1,>W2:B2> \) indicates the field is longer than 1 word beginning at word \( W1 \), bit position \( B1 \) and going through word \( W2 \) bit position \( B2 \).

This type of information can be very helpful when interfacing with other languages such as COMPASS or FORTRAN and also when trying to minimize the size of a record by rearrangement of its fields.

Sincerely,

Ricky W. Butler
Systems Programming
SDC-Integrated Services, Inc.

for

NASA, Langley Research Center
Hampton, Virginia
MS 157B

P.S. To obtain more information or the update mods for this option contact:

Rudeen S. Smith
MS 125A
NASA/Langley Research Center
Hampton, Virginia 23665
(804) 827-2886
Dear Rick:

Since the last time I wrote to PUGN (PUGN #11 - February 1978), many things have happened both here at KU and with Pascal on Honeywell/GCOS. I'll start off with the new happenings with Honeywell Pascal (under PROGRAM L). Things have happened both here at KU and with Pascal on Honeywell/GCOS. There are two major extensions: and "else" clause in the case statement and the variant record, and a relaxation of the type checking when applied to variables and constants of "packed array of char" (the first elements of each are made to align and the shorter is logically blank extended for compares and assignments; strings can be read using read). Pascal is available through Honeywell marketing, but was written and is maintained at the University of Waterloo. Anyone interested in obtaining a copy of the documentation should write to:

The Broadway Bookstore / Kansas Union / The University of Kansas / Lawrence, Kansas 66045 and request a copy of "Pascal on the Honeywell Computer System" ($3.00 plus $.00 postage).

I have been promoting Pascal in the Honeywell Large System Users Association (HLSUA). I am the chairman of the Scientific Language committee and have given 3 talks about Pascal over the last 2 years; one a tutorial about Pascal, and the other 2 comparisons of Pascal compiler and run times versus FORTRAN, B and C (unfortunately Pascal came out on the short end most of the time). I will include a copy of the "comparison" paper with this letter.

Pascal has been in use at the University of Kansas since 1976. Almost all the undergraduate computer science classes use Pascal. We teach a university wide service course which serves as an introduction to programming to over 900 students a semester. For the past two years some portion (at least 1/3) of these students were taught Pascal (the others were taught FORTRAN). This coming Fall semester, the Pascal portion will be slightly greater than a half. Myself, another graduate student, and a faculty member have put together a brochure which we are distributing to the faculty of other schools within the university who use our introductory class. The purpose of the brochure is to introduce the other faculty members to Pascal and to explain why we (CS) want to teach Pascal, instead of FORTRAN, in the introductory course. After sending the brochure, we meet with the faculty from the other department or school and answer any questions they want to ask and further expand upon the reasons for teaching Pascal outlined in the brochure. We have only met with faculty from the School of Engineering. We have had some success. If they can find more credit hour in the majors involved, they have tentatively agreed to allow their students to take Pascal as their first language if we also offer a 1 hour course for their students in which they would learn FORTRAN. We currently have plans to meet with the faculties of Business and Journalism next fall.

If any other schools have done this, I would very much appreciate hearing from you. If anyone is interested in our brochure or in talking about our experiences, I'd be happy to do whatever I can.

Other Pascal news from KU: we have a student oriented Pascal syntax checker (written in B using TACC - probably not portable except to another Honeywell). The syntax checker runs much faster than the compiler and generates much more explicative error messages. It explicitly looks for many of the mistakes commonly made by novice programmers and diagnoses them. There should be a paper written on this project by Jin Hoch and Uwe Pleban in the coming months. I have ported the Path Pascal compiler (written at an University of Illinois and acquired through Dr. Edwin Foudrati at NASA-Langley) to the Honeywell and am currently porting a newer version of the compiler (we have to change 112 out of 7562 lines in the source). We have almost all of the programs that have appeared in PUGN up and running, most of which required only minor changes. (The portability of Pascal and its availability on micro computers have been the most important arguments to others in convincing them of the value of Pascal, let's keep it standard!)

I'd like to thank everyone at PUG central (Andy, Rick, and all the others whom I don't know) for the great job you're doing. PUGN is a tremendous help in promoting Pascal and the standards efforts by PUG-USA and Tony Addyman with BSI are extremely important to the vitality Pascal currently enjoys. Again, thanks.

Sincerely,

Gregory F. Wetzel
Assistant Instructor
Dear Dr. Addyman:

This is a comment on the proposed Pascal standard.

It is good to see that conformant array parameters are to be included in the Pascal standard in a neat and carefully considered manner. This will prevent the proliferation of non-standard implementations (an alarming thought).

I do wish to take issue with the proposal to exclude the "packed" attribute from the conformant array schema (Pascal News 17, p. 54). My reasoning is this.

1. A problem with Pascal perceived by a number of applications programmers is the difficulty of manipulating strings and of formatting text output (and interpreting printable input).

2. The logical response is to make available a library written in standard Pascal which will perform formatting and string manipulation. (Some can be found in Pascal News 17.)

3. If conformant packed arrays are not permitted, such a library must use standard length strings, longer than the longest actual string which is to be processed. Alternatively strings must be processed in unpacked arrays. In either case, there is a wastage of storage space, which is a significant problem for some users. Or, space can be allocated dynamically in chunks for strings. This complicates the library routines, resulting in a wastage of program storage, again a significant problem.

4. The problems cited by A.J. Sale which lead him to recommend against packed conformant arrays are really no more serious than the implementation of packed arrays themselves. When referencing any packed array, information on the bit-length of the component type is always needed. When the packed array is a conformant packed array of conformant packed arrays, the bit length will have to be passed by the calling procedure, rather than being a constant. Since the array dimensions already must be passed, this is hardly a serious problem.

5. More generally, packed arrays should be permitted to be used anywhere that unpacked arrays are permitted, unless there is a very powerful reason to forbid that use. One place where there is a real problem is in the use of a component of a packed array as a variable argument to a procedure. That is the only place where packed arrays are limited, at present. If more limitations are introduced, the result, as Sale suggests, will be non-standard compilers which support conformant packed arrays. This will have a detrimental effect on portability.

My reasoning may appear highly dependent on the perceived need for easy string manipulation facilities. But articles too numerous to mention have been appearing on the topic of strings, and the reason is that this is a problem which is encountered by virtually every applications programmer. So please - let's not go halfway on the conformant array problem.

Thank you for considering my comments.

Yours truly,

Jack Dodds

cc A. J. Sale
J. Miner
Pascal News
Dear Rick:

This letter is to inform you and all PUG members of the introduction of a Pascal-based real-time applications programming language called Micro Concurrent Pascal (mCP). mCP was developed and has been used by ENERTEC over the past two years. ENERTEC is a small systems software house which uses and develops Pascal-based software tools for our programming needs.

Micro Concurrent Pascal was developed from Per Brinch Hansen's Concurrent Pascal; however mCP is a language in its own right. The mCP compiler is a stand-alone program and interpreter/kernels presently exist for the Z80 and 8080/8085 microprocessors.

Brinch Hansen's Concurrent Pascal extends Pascal with the real-time programming constructs called processes, monitors and classes. In addition to the process, monitor and class constructs, Micro Concurrent Pascal contains the device monitor construct.

A device monitor is a variant of a monitor which permits the writing of device drivers directly in mCP. Each device driver is associated with a specific interrupt. Processes call device monitors to do I/O. The DOID statement, permissible only in a device monitor, blocks the process which called the device driver until the associated interrupt occurs. Other statements restricted to device monitors allow an mCP program to access absolute hardware addresses and perform bit manipulations on data. Among other ENERTEC additions are:

- a drop-to-assembly language capability
- separate data types for 8 and 16 bit integers
- string manipulation intrinsic routines
- hexadecimal constants

Additionally, P-code output by the Micro Concurrent Pascal compiler is approximately one third the size of the P-code output by Brinch Hansen's Concurrent Pascal compiler.

I've enclosed a technical article which walks through the programming of a simple real-time operating system in Micro Concurrent Pascal. Anyone interested in mCP is invited to call or write to ENERTEC.

Keep up the great work with Pascal!

Sincerely,

Cynthia Fulton

CF/cc enc.

PASCAL USERS' GROUP

Gentlemen:

I am a deputy district attorney in a rural area at the foot of the Rocky Mountains. The Institute for Law and Research, Washington, D.C., has implemented a Prosecution Management Information System (PRINIS) in COBOL for big machines and for microcomputers.

I am interested in adapting at least part of that system to microcomputers, especially in view of the availability of 8" hard disc drives. Pascal may be the ideal language for it. Can any of your readers provide insights into the process of creating data base management systems with Pascal, and with practical, if not optimum, algorithms for using hard disc storage? I'm fluent in NBASIC and the CP/M systems, but Pascal is new to me. I would appreciate hearing from anyone interested in the PROMIS project, as well as anyone who can recommend books or articles for the study of Pascal. The Pascal available to me presently is the UCSD Pascal for microcomputers.

Finally, I would be interested in comments concerning the relative strengths and weaknesses of the Microcomputer COBOLs for data base management vis-a-vis Pascal (assuming a Pascal implementation which includes random disc files, and reasonable interactive facilities for on-line terminal I/O).

Thank you. I look forward to seeing my first copy of the newsletter.

Sincerely,

Dennis B. Faulk
311 Harrison Ave.
Canon City, CO 81212
(303) 275-1097
Dear Rick:

I am enclosing with this letter notices of two new projects of which I am very excited: the UCSD Pascal Users' Group and SOFTDOC, a medical software network featuring Pascal as the preferred language.

Fundamentally, the reason behind the UCSD users' group is that, to date, it is the best Pascal system for microcomputers, trading somewhat slower execution for speedy disk access (three times faster than CP/M), a superb development and operating system, and compact code, allowing macro programs in mini memories. As we all recognize, because Pascal is so close to the machine, there is a great need to develop a library of commonly used routines so we don't have to continually "reinvent the wheel" each time we program. I and my friends have been using the UCSD system a great deal, and a fair amount of software is beginning to be exchanged -- enough to fill up two volumes. I have included the two Pascal formatters/prettyprinters published in the Pascal News No. 13, as well, and plan to enter the other superb Pascal software tools you publish as time permits.

We microcomputer users receive little benefit from software offered on 9-track tapes (I suspect the tape drive costs more than my entire system); so machine-readable software must be shared on floppy disks. Because UCSD has been so careful (almost paranoid) about preserving the integrity of their RT-11-like disk and directory format, it turns out that anyone running UCSD Pascal on a system with access to an 8-inch floppy drive can share software inexpensively, regardless of the host CPU.

I do have a question about software published in the Pascal News. Programs published in magazines or journals are generally considered to be in the public domain. Would the members of the Pascal User's Group have any objection to my offering, as inexpensively as possible, the software published in the Pascal News to anyone who can utilize software inexpensively, regardless of the host CPU.

SOFTDOC is more ambitious than the users group project. Medical computing has been at an impasse almost since its inception: medically trained people tend not to use tools developed by nonmedical personnel, including programmers, because these tools rarely fit into the peculiarities of medical thinking and practice. So there is a history of failure, and not a little bitterness on the part of computer professionals. Few accepted uses of computers in the health sciences exist outside of the laboratory.

As you can see in the enclosed material, the aim of SOFTDOC is to form a network of health care professionals, via a floppy-disk journal, so that together we can develop medical applications for computers that are truly valued by clinicians. I am informing the members of the PUG of SOFTDOC because UCSD Pascal is the preferred language for programs submitted to SOFTDOC for disk publication. In addition, I believe the enormous potential of Pascal for medical computing (exclusive of applications requiring sizeable mathematical power and speed) has been insufficiently emphasized.

I would be interested in hearing from anyone with further ideas on sharing microcomputer software inexpensively, especially in the area of medical computing. Let me know, too, if you would like to work out some sort of reciprocal sharing arrangement. Perhaps I would send the PUG a copy of each disk as it was released, and you would publish items of interest to the broader PUG.

Sincerely,

Jim Gagne, M.D.
President

SOFTDOC is a new service recently announced by Datamed Research to aid health professionals who are interested in utilizing computer systems in their practices.

Small computers have the potential to serve a myriad of needs in health care practices. Such applications as obtaining the routine portions of histories directly from patients, patient education, and limited assistance with diagnosis or treatment are readily achievable. To date, most authors of medical computer programs have not taken into account the true needs of health care professionals, and the programs have not been utilized by those they were designed to serve. Effective medical computing requires a network of health professionals writing programs and sharing their software.

In the past fifteen years, over a hundred health professional office business systems have reached the market. While the majority have failed, a few have transformed the business office into a streamlined, highly accurate system. Unfortunately, for the small office, the cost of the better systems usually exceeds $30,000. Now, however, with the advent of quality hardware systems for well under $10,000, new, less expensive medical business packages are being released. The difficulty is to locate software of quality amid a rain of inadequate programs.

SOFTDOC will support the emergence of high-quality, low-cost medical computing in the following manner:

1) We are now issuing a call for health-related software to be published in a quarterly machine-readable software journal.

2) The journal will also contain in-depth user reviews of both SOFTDOC and commercial software, so that together we can determine just which programs are the most effective and why.

3) Datamed Research will collect and evaluate vendor's descriptions of commercial software. In addition, user evaluations of software will be collated and summarized. Our findings will be published semiannually in the SOFTDOC journal. Vendors and users who participate in the evaluation will also receive a summary of the findings. Access to software produced in recent years by the American business professionals has been the business office, our initial concentration will be in this area.

The preferred medium of SOFTDOC is IBM-compatible floppy disks; for the convenience of those without 8-inch floppy drives, it will also be issued in printed form. Material on a disk may be submitted to SOFTDOC for inclusion in the first issue until May 1, 1980; all programs must be in source code form and contain adequate documentation. Publication will take place on June 1, 1980, and quarterly thereafter. Subscriptions will cost $55 per year, or $18 per individual disquette. Those who donate software, reviews or articles will receive a one-issue credit per item published.

Subscribers must indicate which they prefer: 8-inch, single-density, single-sided, IBM-compatible floppy disk available in CP/M or UCSD Pascal format (specific) or hard copy. We would like to find someone who can copy the material on 5-1/2 inch diskettes for distribution in that format. However, these are not available at the present.

If you are interested in promoting valid medical uses for microcomputers, we invite you to send us programs you have written. Your software will be given the widest possible distribution. Together, we may change the long overdue promise of medical computing to a reality.
A New, Minimal-Cost Software Club for Users of UCSD Pascal

Introduction.

The UCSD Pascal language system is one of the most sophisticated microcomputer software systems available today. Because of the ease with which one can write and maintain high quality programs of most types, from systems software to business applications to games, it promises to be the vanguard of an enormous interest in Pascal in the coming decade. Already a number of other Pascal implementations have appeared for microprocessors, though none so complete.

UCSD Pascal compiles its programs to P-code, designed for a hypothetical 16-bit stack machine that must be emulated in software on most microprocessors. As a result, once the P-code interpreter has been installed, programs written in UCSD Pascal may run on any microprocessor without modification. Even the disk formats are the same, except for the mini-floppies used for the Apple, North Star, or TRS-80. So disk software in either source or object form may be freely shared among users of such diverse microprocessors as a PDP-11 or an 8080.

The Pascal Users Group.

It would seem natural for a large users group to arise to share software. To date, however, only the original Pascal Users Group ("PUG") serves this function. Primarily, they support the standard language based on the Jensen and Wirth Pascal User Manual and report and on available Pascal implementations and programmer opportunities. Only secondarily does the PUG disseminate software (based on Jensen and Wirth Pascal), although since 1978 the PUG has published several superb "software tools". The major difficulty with the PUG newsletter is that it is offered only on paper; any machine-readable software is offered on 9-track tapes, which are not supported by the majority of microcomputers. So a microcomputer user must type the software into the machine on his/her own.

A UCSD Pascal Users Group on machine-readable media.

Datamed Research is announcing the formation of a UCSD Pascal users' group. It will take a form very similar to the highly respected CP/M Users Group; all offerings will be on 8-inch, single density, IBM-compatible soft-sectored floppy, offered virtually at cost ($10 per disk). Software will be donated by interested users. Software donors will receive a free disk volume of their choice in acknowledgement of their donation. For software to be accepted for distribution it MUST come with adequate documentation on the disk. Further, with rare exceptions it must be supplied in source code to allow other users to adapt it to their systems.

Potential sources of Pascal software abound; by no means must one donate only original work. There is a mountain of public-domain Basic software that is easily adapted to Pascal. In the process, one can usually spruce up the program a good deal, because Pascal is so much easier to work with than Basic. It will be important, in addition, for the users to begin a library of Pascal procedures and functions to handle the more common programming problems. For example, we need a set of mathematical functions for complex variables, statistical functions, and basic business software support (routines to translate integers into dollars and cents and vice versa) to realize the full power of the language.

You can find out more about the present status of the users group by sending a self-addressed, stamped envelope to the following address:

DATAMED RESEARCH
1433 Roscomare Road
Los Angeles, CA 90024

Alternatively, 8-inch floppies can be ordered at $10 per volume; there are two volumes available at the present time. Because the BIOS for the 512-byte sectors is written for Digital Research's CP/M-based macroassembler, the second volume contains both a CP/M- and a UCSD-format disk (though if you prefer, both disks can be of the same type; the volume is of use primarily to those who have both CP/M and the UCSD system, however) and costs $20. California residents must add 6% sales tax. Be sure to specify UCSD or CP/M format.
Pascal Standard: Progress Report
by Jim Miner (1980-07-01)

A serious disagreement over conformant array parameters is the only major remaining obstacle to obtaining an ISO standard. Hopefully both sides will quickly resolve this impasse in a friendly and diplomatic way, because there is a real possibility that one or more national groups will be compelled by time constraints to break with the international effort and seek to obtain their own standard.

RECENT EVENTS

Voting on DP 7185

The latest draft standard ("DP 7185") was published in Pascal News #18 and in SIGPLAN Notices (April 1980). Votes cast by specific national bodies on this draft are as follows.

<table>
<thead>
<tr>
<th>Approval</th>
<th>Disapproval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finland</td>
<td>Australia **</td>
</tr>
<tr>
<td></td>
<td>Czechoslovakia *</td>
</tr>
<tr>
<td>Hungary</td>
<td>Denmark *</td>
</tr>
<tr>
<td>Italy</td>
<td>France</td>
</tr>
<tr>
<td>Romania *</td>
<td>Netherlands</td>
</tr>
<tr>
<td>Sweden</td>
<td>U.K.</td>
</tr>
<tr>
<td>* &quot;Observer&quot; member -- vote is advisory. ** Australia has become a &quot;Principal&quot; member since this vote.</td>
<td></td>
</tr>
</tbody>
</table>

Working Group 4 Meeting

The comments accompanying the votes revealed several technical inadequacies as well as some issues on which there is disagreement. Tony Addyman’s report "The Pascal Standard: Progress and Problems" (below) discusses several of these issues.

The ISO Working Group on Pascal (WG4) met in Manchester England during June in an effort to resolve these issues and to prepare a second Draft Proposal. (See Pascal News #17, pages 83-84, regarding the origins of WG4.) Attendees were:

Tony Addyman (U.K.)
Burkhard Austermuehl (Germany)
Albrecht Biedl (Germany)
Cees Bron (Netherlands)
Joe Cointment (U.S.A.)
Christian Craff (France)
Jacques Faré (France)
Charles Haynes (U.S.A.)
Ruth Higgins (U.S.A.)
Mike Istinger (U.S.A.)

Pierre Maurice (France)
Jim Miner (U.S.A.)
Kohei Noshita (Japan)
Bill Price (U.S.A.)
Helmut Sandmayer (Switzerland)
Karl-Heinz Sarges (Germany)
Barry Smith (U.S.A.)
Alain Tisserant (France)
David Williams (Canada)

JPC Meeting

A few days after the Manchester meeting, the U.S.A. committee (JPC) met in Portland Oregon. Out of that meeting came the memos from David Jones to WG4 and to the National Bureau of Standards which are reproduced below.

THE PROBLEM

As Tony’s article points out, the most difficult problem which the standard now faces is the disagreement over "conformant array parameters." It has been clear to many of us who are deeply involved in the standardization work for some time that this topic could give us much trouble. The extent of the present difficulty became more obvious at the Working Group 4 meeting in June. No conclusion was reached by WG4 regarding conformant array parameters.

The papers by Tony Addyman and David Jones, together with Arthur Sale’s article in Pascal News #17 (pages 54-56), provide much insight into the nature of the disagreement.

In favor of conformant arrays

The capability to allow formal array parameters to have "adjustable" index ranges is deemed necessary for the construction of libraries of separately compiled procedures, especially numerical routines. It is argued that failure to standardize now on the form of such a capability will make future standardization impractical due to many incompatible extensions which will be made to provide the capability.

Based on statements made in the WG4 meeting, the following member bodies are likely to vote "No" on a Draft Proposal which does not contain a conformant array feature: Germany, Netherlands, U.K.

Against conformant arrays

Those opposing the inclusion of conformant arrays in the standard argue that the proposal is technically flawed and as a result that its inclusion in the draft will delay the entire standard. (The attachment to David Jones’ memo to Working Group 4 contains a technical assessment of the existing proposal.) It is also argued that conformant arrays are not needed more than other extensions which have not been included in the draft proposal.

Based on statements made in the WG4 meeting, member bodies likely to vote "No" if conformant arrays remain are Canada, Japan, U.S.A.

Variations on the theme

Some member bodies have expressed a preference for generalizations of the conformant array feature; Germany, for example, voted "No" partly because value and packed conformant arrays are not allowed.

The U.S.A., which has expressed opposition to conformant arrays on several occasions, proposed a compromise in its vote. The compromise would make conformant arrays optional for an implementation, but with the requirement that any such capability supported by an implementation have the syntax and semantics specified in the standard. Several members of WG4 expressed dislike of this proposal.

CONCLUSION

The standard has been stalled by the disagreement over conformant array parameters. In order to obtain an ISO standard, it is necessary that a compromise of some kind be reached. At this time it is hard to predict what the nature of that compromise will be.
The Pascal Standard: Progress and Problems,
May, 1980

A. M. Addyman
University of Manchester

1. Introduction

Within the International Standards Organization (ISO), there is a work item which is to result in the production of a standard for the programming language Pascal. This work began in ISO in October 1978 as the result of a proposal from the United Kingdom. Work in the United Kingdom began early in 1977. At the time of writing this report, a ballot is taking place within ISO on the acceptability of the first Draft Proposal for the Pascal standard. This report, written immediately after the April 1980 meeting of the U.S. Joint Pascal Committee (X3J9), contains a summary of the substantial progress made to date and discusses the few remaining problems which stand in the way of international agreement.

2. Progress

There is now agreement on the details of all the main areas, although in one or two areas the wording is being improved or drafting errors are being corrected. The areas in which agreement has been reached include:

- lexical issues,
- scope rules,
- type rules,
- the syntax and semantics of the statements and declarations, almost all of the input and output facilities.

Indeed, since there is agreement on so much, it would be better to devote space to the consideration of those issues which have yet to be resolved. Before doing so it should be noted that there is agreement that a standard is needed without delay. This attitude has helped to resolve minor differences of view, since neither party has wanted to risk the standard on such issues.

3. Problems

The outstanding problems will be divided into two categories - minor and major. The major problems are the ones which could substantially delay the production of the standard. The category into which a problem has been placed is necessarily a matter of personal judgement.

3.1 Minor Problems

3.1.1 Alternative Lexical Tokens

The issue is simply that (and.) should be accepted as alternatives for [ and ]. There are strong feelings both for and against this. The strongest opposition appears to be from the U.K. The probably outcome will be acceptance of the alternative tokens.

3.1.2 String Truncation on writing

This is a request which involves a change from the current de facto definition. Its advocates cite efficiency, utility and frequent violation of the de facto definition as justification for the change. Opponents argue that

(a) this is a change and consequently must be rejected, and
(b) that a truncated representation of the array cannot in any way represent the array.

The possible outcome is unclear, but will undoubtedly be influenced by the U.S.A. position on the major problem (see later).

3.1.3 Tag-fields

There are three loosely related problems in this area:

(a) a change to prohibit use of tag-fields as var-parameters
(b) a relaxation of the syntax to replace "type" by "type-identifier"
(c) a change which would disallow the creation of tag-less variants

Each of these is a change to the de facto definition and as such provoke opposition.

The first is proposed in the interests of promoting the implementation of certain checks desired by the Draft Proposal. It will probably be accepted.

The second change is a change to the syntax to eliminate one of the circumstances in which a type-identifier is necessary and a type definition is unacceptable. The change was strongly opposed at the Pascal Experts meeting in Turin. I expect this opposition to continue.

The third change is proposed on the grounds that its only use are in implementation dependent "dirty tricks". While this is untrue, the wording of the Draft Proposal suggests that an implementation which performs checks in this area will have to provide a tag-field if the programmer does not. The only justification for this feature which is within the proposed standard is associated with the saving of storage space for variables. Since a large number of implementations incorporate this restriction, which is aimed at improving security, there is a possibility that it will be accepted.
3.1.4 New and Dispose

There is a form of these standard procedures which may be used to reduce the storage requirements of a program. The use of this feature may lead to errors which are difficult for the programmer to detect, furthermore an implementation can detect such errors only by using additional storage. There is pressure to have this form of new and dispose removed.

Given the increasing usage of Pascal on microcomputers it is likely that the definition of new will be unchanged. There is a much stronger case for changing dispose since most implementations maintain enough information to ensure the security of the heap. The final irony is that the Draft Proposal identifies two error conditions which can only be detected by maintaining enough information to make this form of dispose redundant.

3.1.5 The Rest

There are a number of minor problems which have been raised by various parties and subsequently dropped e.g. the U.K. Pascal group has expressed a desire to remove pack, unpack and page from the language; other European groups have requested extensions to the case-statement and changes to the syntax of a block etc. There is a danger that decisions to make changes in any of the areas cited above may provoke more requests.

3.2 The Major Problem

3.2.1 Introduction

There appears to be only one substantial problem which may prevent agreement being reached on a Pascal standard. This is the problem of adjustable array parameters.

In the de facto definition of Pascal, a parameter of a procedure must have a specific type which in the case of an array will include a specification of the bounds of the array. This is viewed by many people as an unacceptable restriction in a language that is being proposed for international standardisation. As a result of the comments received on the document ISO/TC97/SC5 N462, the U.K. Pascal group resolved to introduce into the draft a minimal facility which would address the problem. The U.K. solution provided for var-parameters but not value parameters and also excluded packed arrays. The proposal from the U.K. has received objections on two counts:

(a) it is a change to the language - in particular, more work should be done on the details of such a feature before it is added to the language.

(b) the feature is too restrictive - value parameters and/or packed arrays should also be allowed.

To clarify matters the arguments which support the three positions will be presented separately.

3.2.2 In favour of the Draft Proposal

1. There is great demand for the feature to be added to the language, and those making the demands have not specified any particular syntax or semantics. Those supporting the addition include Prof. Hoare and Prof. Wirth.

2. In the interests of portability the feature should be required in any implementation of a Pascal processor.

3. There are no technical difficulties with implementing the feature in the Draft Proposal since all the "run-time" operations that are required already exist.

4. Requiring value adjustable array parameters has an impact on the procedure calling mechanism - the amount of space required by a procedure cannot always be determined at compile-time. There is concern that there may be existing implementations which rely on such a determination at compile-time and which would therefore be destroyed by the introduction of value adjustable array parameters.

5. Requiring packed adjustable array parameters places increased overheads on an implementation which packs multidimensional arrays. Such overheads may result in a reduction in the extent to which a packing request is heeded.

6. If action is not taken at this time a number of vendors will surely introduce incompatible extensions to fulfill this obvious need. Such action would effectively prevent future standardisation of this feature.

7. Of all the requests for extensions received during the comment period on ISO/TC97/SC5 N462, this is the only one which adds to the functionality of the language. All the other requests addressed issues of convenience and/or efficiency.

3.2.3 In favour of a less restrictive proposal

1. All the above arguments are accepted apart from 4 and 5.

2. Those in favour of value adjustable array parameters claim that no existing implementations will be embarrassed and claim (correctly) that there are no technical problems.

3. Those in favour of packing fall into two distinct groups:

(a) those who believe that there are no implementation problems and that in the interests of generality the restriction should be removed.

(b) those who wish to use string constants as actual parameters. They appear to need both value (since a constant is not permitted as an actual var-parameter) and packed (since the Draft Proposal specifies that string constants are of a packed type). An alternative solution to this problem is to change the specification so that the type of string constant is context dependent (as in the case for set-constructors) in which case a string constant could also be a constant of an unpacked type. The same proposal also requires that those operators which apply to packed character arrays also apply to unpacked character arrays. This has the considerable merit of removing the only case in which
the prefix "packed" is used for reasons other than storage reduction.

3.2.4 In favour of the feature being optional

This is a view expressed by the U.S.A. Pascal committee (X3J9).

1. A language designer must not add to a language any feature that is not very well understood, that has not been implemented, or that has not been used in real programs. The proposed adjustable array parameter feature is just such a feature. This feature should be widely implemented and used before it is incorporated into a standard for Pascal.

2. By placing the proposal in an appendix entitled "Recommended Extension" we derive the benefit of having the opportunity to implement the feature before casting it in concrete.

3. Implementors who add a feature which performs this function are required to comply with the recommended extension. This will make compatibility with any future extended Pascal more likely without foregoing the possibility of learning more about the feature in the interim.

3.2.5 The Probable Outcome

There is considerable pressure from several ISO member bodies (the U.K. excepted) to remove the restrictions which the Draft Proposal incorporates relating to adjustable array parameters. The probable conclusion will be to permit value but prohibit packed and at the same time introduce the changes described above relating to the operations etc. available for character arrays. Unfortunately the proposal from the U.S.A. for removal of the feature to an appendix is likely to be opposed strongly by one or more member bodies. This view is based on the comments received from other ISO member bodies since the April X3J9 meeting. The strength of support for removal of the restrictions is unlikely to be compatible everywhere with a willingness to accept less than is contained in the Draft Proposal. One possible solution would be for X3J9 to accept the feature as part of the language. At this stage this does not seem likely since the X3J9 position was taken for largely non-technical reasons. This observation is justified as follows:

1. X3J9 is requesting changes to the existing de facto definition while objecting to this extension.

2. X3J9 is currently soliciting extension proposals - it is unlikely that any such proposals will be acceptable by their criteria in 3.2.4.1 above.

3. To promote portability and improve the probability of agreement in a future standard, the extension must be implemented as specified in the appendix. An implementor may only experiment with an alternative if the recommended extension is also implemented. This adds no new freedom to the implementor since language extensions are not prohibited by the Draft Proposal!

4. X3J9 also supports the removal of some of the restrictions mentioned earlier.

3.3 Conclusions

The meeting of ISO/TC97/SC5/WG4(Pascal) to be held in June 1980 will be a crucial one. There is pressure within the United States to move on to consideration of extensions - this is being delayed by the current activities. In the United Kingdom there is a government funded project to create a validating mechanism for Pascal. This clearly needs a standard to validate against. Significant progress is required on this project by April 1981!

A negative vote by any member body on the second Draft Proposal, later this year, will probably terminate the international standardization effort because it will introduce delays which are unacceptable to one or more member bodies who will have little alternative but to produce national standards instead.

There is a real danger that one of more ISO member bodies will find the removal of adjustable array parameters to an appendix as unacceptable as the United States finds their inclusion in the body of the standard.
27 June 1980

MEMO

To: ISO/TC97/SC5/WG4
Re: U. S. concerns on Pascal Standardization With Respect to the Conformant-array Extensions

The Joint X3J9/IEEE Pascal Standards Committee has resolved to express its concern that the issue of conformant array parameters may significantly delay the acceptance of the draft proposed standard for Pascal as an international standard. The committee is anxious to explore any option which will lead to a solution of the conflict over this issue acceptable to all member bodies of SC5.

As you know, the US member body of ISO TC97/SC 5 voted against the acceptance of the first draft proposal, on the grounds that the conformant array feature should be described in an appendix to the standard. This position was a compromise offered in the hope that it would be acceptable to the other member bodies of SC 5 and thereby an international consensus could be quickly achieved. The position did not, in fact, reflect the true sentiment of the JPC, as expressed in a number of formal and informal votes, which was that a conformant array feature should not be included in the current standard for Pascal. In the beginning there was no proposal available to evaluate technically, and the committee's view was based on strategic considerations. These were that the introduction of a new and largely untried feature at such a late date would introduce technical problems which could not be resolved in time to avoid delaying the acceptance of the standard. This has in fact turned out to be the case, since the first proposal for a conformant array feature was sufficiently technically flawed to justify its replacement by a quite different proposal. There are still major technical objections to the latter so that the view of the JPC on conformant arrays remains unchanged, although it is now based on technical considerations. These are described in the attachment (which was accepted unanimously).

U. S. concerns on Pascal Standardization

This committee understands and shares the view that the conformant array feature attempts to solve a significant technical deficiency in Pascal. However, it feels that the technical objections should be resolved before such a feature is included in an International or American National Standard. The committee believes that this leaves two possible courses of action if a failure to agree on an International standard is to be avoided. The first is that a major international effort through the Working Group must be mounted to prepare a technically sound proposal. The committee believes that this is likely to require yet another complete revision of the proposed feature. Sufficient time must be made available for such work to be completed and properly evaluated. The second approach is that we should proceed as quickly as possible to standardize the language at a level at which it has been widely used for a number of years.

It is clear that the second offers the quickest route to a standard and we strongly recommend that it be adopted. However, we further recommend that the effort identified in the first approach be simultaneously initiated and that an acceptable conformant array proposal should be defined and included in a subsequent standard for Pascal as soon as possible.

Yours sincerely,

D. T. Jones
International Representative
Joint ANSI/X3J9 - IEEE Pascal Committee

Enclosure
1.0 Overview and general problems

The U.S. Joint Pascal Committee (X3J9) created an ad hoc task group to investigate the conformant array extension appearing in JPC/80-161 (Working draft/6) (6.6.3.1). This report, together with JPC international liaison David Jones' cover letter to the international working group (WG4), is the result of the task group's investigation. Contributing members of the task group included Bob Dietrich, Hellmut Golde, Steve Hiebert, Ruth Higgins, Al Hoffman, Leslie Klein, Bob Lange, Jim Miner, Bill Price, Sam Roberts, Tom Rudkin, Larry Weber (chairperson), and Tom Wilcox.

1.1 Lack of implementation experience

The current proposal has no widely known implementations. Various portions of the extension have been implemented in different compilers, but the group of features proposed here have never been combined together, except on paper. Furthermore, the implementations of the various parts of the extension have not (of course!) been in the context of the proposed standard. Since this is a new feature to the language, the introduction of this extension in the standard document is especially distressing.

1.2 Large change to text of standard

The conformant-array extension requires a large amount of text in the standard in order to describe it. Moreover, it requires modifications to sections outside of section 6.6.3 on parameters. In other words, the extension interacts -- at least in its description -- with many other parts of the language. For example, in section 6.7.1 the alternative "bound-identifier" has to be added.

This means that the extension is major, with wide impact on the language. This is especially unfortunate in view of the fact that it only provides a single capability -- that of array parameters with adjustable bounds. A broader capability, might not require a significantly larger description.

1.3 Conformant-array concept not defined

It is of the essence of the Pascal language, and its principal distinguishing characteristic, that it is "based on certain fundamental concepts clearly and naturally reflected by the language" (page 1, section 0, forward to the Draft ISO/DP 7185). It is difficult, at best, to identify a fundamental concept that this extension is to support. The best approximation yet suggested is the adjustability of the bounds of a scalar-type used as the index-type of an array-type under certain circumstances of parameter usage. Inasmuch as this concept is founded on at least five identifiable concepts, it is difficult to see how it may be considered fundamental.

This absence of fundamental underlying abstraction is foreign to the nature of the language. This absence leads inexorably to user confusion and to language-designer confusion. The user is not provided a concept on which to base his understanding; the designer, likewise, is given no guidance in his language design. Since user experience is lacking no evidence exists from which to draw any conclusions with respect to the lack of user understandability. However, the lack of guidance to the language designer is quite nicely evident from the volume of technical objection; the most acute examples are the dilemmas of packing and of value-parameters.

2.0 Problems with existing proposal

2.1 Set of types that may have to conform is unrestricted

The conformant-array extension provides no way to identify, at the point of declaration, the array types that may have to conform to some conformant-array parameter. Consequently, an implementation must ensure, a priori, that ALL array types can be handled correctly by the implementation of the conformant-array parameter extension. Hence, a user may have to endure severe implementation inefficiencies even though he does not use the conformant-array parameter extension. For example, an implementation of packed conformant-array parameters (an almost irresistible evolution of the present extension) may make many of the possible forms of data packing totally impractical. A solution that is integrated with the type naming mechanism would alleviate this problem.

2.2 Structural Compatibility

One of the fundamental clarifying decisions made in developing the draft standard from Jensen and Wirth was the rejection of so-called "structural type-compatibility" in favor of the more natural "name compatibility" (or a variation thereon). Such decisions have had a profound effect on the resulting language; it is important that such principles be applied consistently throughout the language.

Unfortunately, two areas of the existing (Jensen and Wirth) language resisted consistent application of "name compatibility": set-types and string-types. Both of these
problems are directly attributable to the existence of inadequately typed value designators (i.e., character-string constants and set-constructors). It was deemed necessary to violate "name compatibility" in these two cases in order to avoid introducing new (and incompatible) language features.

The conformant-array extensions introduced in N510 and in DP 7185 both violate the underlying principle of "name-compatibility"; we have seen no attempt to justify this violation. This is inexcusable in the absence of problems of upward-compatibility, very simply because conformant-arrays are an extension.

One practical effect of this unnatural regression to structural-compatibility, as discussed elsewhere in more detail, is the difficulty encountered in extending the conformant-array capability to allow multi-dimensional packed arrays.

2.3 Parameter List Congruency

In the comments from the French member body (p.3, 6.6.3.6), they note that "the parameter lists (x,y:t) and (x:t, y:t) seem to be not congruent" and that this is the only part of the language where these two notations are not entirely equivalent. It is a correct observation that these are not congruent. However, given the current form of the conformant-array proposal, this surprising and aesthetically unpleasant inconsistency is absolutely necessary. If the two parameter list forms were congruent (as in N510), then the following example would be a legal program fragment:

```pascal
type t = integer;

proc p1(var fl,f2: array[i..j: t] of u);
begin fl:= f2 end; [end - p1]

proc p2 (proc fp(var fl: array[i1..j1:t] of u;
    var f2: array[i2..j2:t] of u));
var a: array[l..2] of v;
    var b: array[l..3] of v;
begin fp(a,b) end; [end - p2]
begin p2(p1) end;
```

It is impossible to know at compile time that the assignment (fl:= f2) is an error. To remove the necessity of this run-time check, a seemingly unrelated aspect of the language had to be altered. The alteration has been recognized as undesirable and the reason for it was certainly not obvious. It took some time to detect the effect of conformant-array-schemas on parameter-list congruency. In addition, there may be other apparently unrelated aspects that, as yet, have not been discovered.

2.4 Need to name a conformant array schema

There is no construct to allow the use of an identifier to denote a conformant array schema:

```
TYPE varray = array[i..j: integer] of integer;

PROCEDURE p(var param: varray);
```

The lack of this construct makes the proposed conformant array schema weaker, due to considerations of consistency and user convenience.

Before proceeding, it must be noted that the naming construct above must be accompanied some means of distinguishing the array bounds "([i..j])" for each individual usage. It is not clear that the currently proposed conformant array extension allows such a capability; this is a general problem in itself as well as a limitation on extensability (see section 3.5).

The first objection to the proposed conformant array extension is the bulkiness of the construct. The parameter list of a procedure or function is frequently placed on one line. The use of a conformant array schema makes this virtually impossible when more than one parameter exists. This and the added user cost of retyping the schema become significant when the same schema is used over and over again, as, say, in a library of mathematical routines.

When one conformant array uses another, in the following manner, the lack of an identifier becomes a clear oversight in the language:

```pascal
procedure p(var a: array[lowa..higha: atype]
    of arecord;
    var b,c: array[low..xhigh: integer;
        clow..chigh: color] of array [lowa2..higha2: atype]
        of arecord);
```

Here it is desirable that the type of "a" in the type of the components of "b" and "c" to be the same.

The unfortunate consequence of adding the inadequately conceived conformant array schema to Pascal is a reduction in the prime desirabilities of convenience of usage and clarity of the printed program.
The lack of an identifier construct for conformant array schemas results in user, language, and implementation inconsistencies. Except for procedure and function parameters, the conformant array schema is the only construct in the parameter list that is not a single word. To new students of the language, it will always appear inconsistent. And, since the parsing of conformant array schemas is so different from other parameter-type-identifiers, it becomes an exception case, resulting in added complexity in the compiler.

The proposed conformant array schema is also shortsighted in that it does not permit the use of a conformant array schema as a part of a record, to be passed as a parameter. For example, many programs make use of dynamic "strings" implemented as records, i.e.

```pascal
type string = record
  length: 0..80;
  chars: array[1..80] of char
end;
```

for a dynamic "string" of maximum length 80. Supposing it were necessary to write a string-handling routine to handle records with differing maximum lengths, one could, with the help of a schema label, construct the following:

```pascal
type natural = 1..maxint;
dynamicarray = array[i..j: natural] of char;
string = record
  length: integer;
  chars: dynamicarray
end;
```

One might therefore argue that for uniformity and possibly as an aid in compiler error recovery, the character "," should be used in the conformant-array extension.

However, there is unresolved disagreement as to whether the separator should be a comma or a semicolon. The existence of this disagreement demonstrates that the nature of the object to be separated is not well understood nor well specified.

2.5 Required Runtime checking of types

The proposed scheme specifies that the type of the formal parameter is the same as the type of the actual parameter. This presents serious difficulties when a conformant parameter is further used as an actual parameter, as illustrated in the following example.

```pascal
program example;
  type arraytype = array[1..10] of integer;
  var
    a : arraytype;
    b : array[1..10] of integer;
    c : array[1..11] of integer;
  procedure simplearray (var a:arraytype);
    begin
  end;
  procedure fancyarray(var a:array[m..n:integer] of integer);
    begin
      simplearray(a)
    end;
begin {main program}
  fancyarray(a) ; {legal}
  fancyarray(b) ; {illegal - name incompatible}
  fancyarray(c) ; {illegal - structure incompatible}
end.
```

array[u..v:T1; j..k:T2] of T3
to be equivalent to

array[u..v:T1] of array[j..k:T2] of T3.

This conflicts with the use of the character "," to express a similar equivalence for array types (6.4.3.2), where

array [T4, T5] of T6

is equivalent to


One might therefore argue that for uniformity and possibly as an aid in compiler error recovery, the character "," should be used in the conformant-array extension.

2.4 Separator ";"

The abbreviated form for contained conformant-array-schemata introduces the character ";" as an abbreviation for the sequence "]" "of" "array" "]" (6.6.3.1), thus allowing the form

array[u..v:T1; j..k:T2] of T3
Another illustration of runtime type checking is shown in the following example.

type
  natural = 0..maxint;
procedure pl(var b:array[i .. j:natural] of u);
begin
begin pl(a) end;
end;

In this example, the passing of the variable "a" to "pl" may or may not be valid, depending on the actual parameter passed to "p2".

This problem is not addressed by the UK Member Body comments on DP 7815.

3.0 Limitations of existing proposal

The following items are brief descriptions of features that could someday be considered as possible extensions to the language. An evaluation and rationale for their desirability has not been completed at this time. The process of including these is impacted by the current definition of the conformant array extension. It is felt that unifying fundamental abstractions must be developed to cover the total set of any newly defined features.

3.1 Leading index types

Only leading index types of conformant-array-schemata are adjustable. Thus,

array[j+.k:T1] of array[T2] of T3

is acceptable, while

array[T2] of array[j+.k:T1] of T3

is not (6.6.3.1). This introduces an asymmetry into the definition. While a relaxation of this restriction does not offer any additional functionality, it would allow a more natural expression of certain relationships between index types.

3.2 The lack of packing

The conformant-array extension, as defined in Working Draft/6, restricts the allowable actual parameters to arrays not having the attribute "packed". This restriction eliminates the direct use of conformant arrays for string handling under the current limitation that the only arrays of char-type that may be compared, written to files or declared as constants are those arrays having the attribute "packed". This particular problem could be corrected by removing the "packed" restriction on string type although care would still be required on the part of the programmer to use only arrays with lower bounds of one and run-time checks would be required to ensure this care had been taken. Even if this string-type problem were resolved, the lack of orthogonality contradicts the Jensen-Wirth Report in which the obvious intent is that packed and unpacked arrays be generally equivalent except for the possible differences in storage requirements.

3.3 Value conformant-arrays

Introduction of a value parameter as part of the conformant-array extension is a natural addition, and there seems to be good reasons to consider this aspect of the conformant-array parameter. However, if this feature were to be added to the extension, then this is the first instance of a case where the size of the activation record is not known during compilation. The unknown size of the activation record causes a problem in an implementation that relies on knowing the activation record size in order to handle activation stack overflow. This is not to say that efficient implementations are impossible, but the two situations must be treated efficiently by compilers.

3.4 Conformant-arrays and bounds limitations

The conformant-array extension is not sufficiently general nor extensible: it does not provide the ability to fix either the lower or upper bound of a given index specification. Nor does it allow the user to equate the extent of one index specification with the extent of another, be it within the same conformant-array parameter or a different conformant-array parameter. This deficiency results in increased time and space complexity and hinders compiler optimization. Moreover, it requires an author to either validate one or more conditions or trust the caller. The former introduces further deterioration of efficiency while the latter is inconsistent with the strongly-typed nature of Pascal. In addition, this lack in the conformant-array extension is in conflict with one of its primary uses: the construction of independent array manipulation routines. For example, possible uses of conformant-array parameters include general matrix multiplication and inversion routines where one would like to place restrictions on the bounds and interrelationship
between index types of the actual parameters.

3.5 Conformant scalar-types

The conformant-array extension addresses only the role of a scalar-type as an index-type of an array-type parameter. It ignores the many other roles where it is desirable to conform a scalar-type parameter. A few such roles where such conformance might be desirable are:

1. as the type of a parameter;
2. as the base type of a set;
3. as the component type of an array;
4. as the type of a field;
5. as the index-type of an array used as the type of a field.

TO: National Bureau of Standards
FROM: David Jones
X3J9 International Liaison
SUBJECT: Report by A.M. Addyman

The Joint ANSI/X3J9 - IEEE Pascal Standards Committee (JPC) has received a copy of a report, "The Pascal Standard: Progress and Problems," written by A.M. Addyman of the University of Manchester. This report, hereafter referred to as JPC/80-164, presents an interpretation of the current impasse in the Pascal standardization effort with which JPC does not agree. I have been charged, as the JPC International Liaison, to present the committee's point of view.

The primary issue over which Mr. Addyman and the committee disagree is discussed in sections 3.2.5 and 3.3 of JPC/80-164, although JPC takes issue with remarks in other sections. Before addressing the comments specifically, however, I shall present a summary of JPC's point of view.

The true sentiment of the committee is that a conformant array parameter feature should not be included in the version of Pascal being standardized through the current effort. This view has been repeatedly documented, by both formal and informal resolutions passed either unanimously or by large majorities, beginning with the first time JPC became aware that the BSI group was considering the introduction of this feature. Initially, the opposition was based on strategic grounds (i.e., there was no proposal to formally evaluate). These were that the delay introduced by requiring a technical evaluation prior to acceptance of the feature would substantially postpone the adoption of a standard. The JPC does believe that the conformant array extension attempts to solve a real problem that will have to be eventually solved, and that finding such a solution is a matter of urgency.

The pessimism of JPC was justified in that the initial proposal offered by BSI was so flawed that it was withdrawn and replaced by an entirely new proposal at the Experts Group Meeting in Turin in November 1979. It is the position of JPC that this second proposal still contains technical errors and deficiencies sufficiently grave that yet another complete revision of the proposal will probably be required before an acceptable solution.
to the problem is found. Consequently, the strategic objections remain, but are now substantiated by technical considerations.

Nevertheless, when the committee voted in April, 1980 to recommend that the U.S. position should be to disapprove the draft proposal identifying conformant array parameters as being the only issue, it only required that this feature be removed to an appendix so that its implementation could be made optional. This represented a major compromise which, from the JPC point of view, was far from the real sentiment requiring that the feature be removed entirely from the proposal.

JPC is convinced that it is in the best interests of the Pascal User Community that any revision or extension to the language be supported by sound technical grounds, and that it is better to take the time to do it correctly or to accept a standard without conformant array parameters than to accept a technically inadequate proposal merely because it is timely to do so.

As far as the actual comments in JPC/80-164 are concerned, the remark in section 3.3.2 on support by Professors Hoare and Wirth should be qualified by the results of the discussions members of JPC had with them before and during the April meeting, of which Mr. Addyman was aware. Both indicated that the U.S. compromise was preferable to delaying the standard, and Professor Hoare himself was the source of this method of introducing this extension. The substitution of the word "standardizer" for "designer" in 3.2.4, paragraph 1, line 1, would accurately reflect the U.S. position. Without the substitution, it does not. Thus 3.2.5, paragraph 2, is also misleading. The use of the term "(correctly)" in 3.2.3, paragraph 2, is difficult to substantiate. The JPC is particularly at odds with the position that non-technical reasons were the justification for its disapproval. We cannot assume Mr. Addyman is referring to our strategic reasons because these reasons have a technical basis. Even in the beginning, the basic issues were technical although they could not yet be identified. Consequently, Mr. Addyman's remark must be construed as implying a political basis for the JPC's position. This is certainly not the case and we disagree with Mr. Addyman's justification for his point of view as expressed in 3.2.5, paragraphs numbered 1 to 4. The following numbered paragraphs discuss our corresponding disagreement:

1. There have been many changes to the de facto definition of Pascal which have not been regarded as extensions and have been the subject of wide implementation and use. This does not apply to the feature in question, reflecting consistency in JPC's position in this regard.

2. It is a subjective opinion that the criteria of 3.2.4, item 1, would preclude other extensions. It is stated quite clearly within the proposed standard that implementation dependent features are allowed, and that by implication a user is free to provide one or more versions of any given feature. By this means, an extension could become widely implemented before acceptance in a standard. In particular, an Appendix could be created for such a feature for the reasons in 3.2.4, paragraph 2, of JPC/80-164.

3. The JPC would prefer that the conformant-array extension be removed entirely from this standard for technical reasons. However, we recognized the claims of the other member bodies that they require this capability in the language. Therefore, the JPC proposed that the extension be in an appendix to address our concerns and we proposed that if the extension were implemented, it was to be implemented in the format specified to encourage acceptance by the other member countries. Since it is our preference to remove the extension entirely, it would be consistent with our position to soften the wording from a requirement to a recommendation.

4. JPC does indeed support the removal of these restrictions, but feels that the technical issues raised by doing so would introduce an unjustifiable delay into the standardization process.

Addressing section 3.3, it is the view of JPC that the position taken by Mr. Addyman (i.e., a negative vote would terminate the standardization process) is unduly pessimistic. In addition, this statement represents unwarranted pressure on the U.S. and the other two countries which voted against the conformant array extension due to significant technical deficiencies.
Implementation Notes

Editor's comments

Well, it was bound to happen. My section of issue #17 got scrambled. The right half of page 88 shouldn't have appeared at all, the Zilog 2-BB reports became recursive, and the machine-dependent section was all out of sequence. My sincerest apologies go to Arthur Sale, whose letter on the Burroughs 8700 implementation was dropped completely, and to my co-editor Greg Marshall, whose hard work on the One-Purpose Coupon went without credit. Things should be straightened out with this issue (I hope).

Just to add to the overall confusion, I've changed my address and phone number within Tektronix. This move is not intended to make it more difficult to reach me. Mail to my old address will be forwarded for the next few years, and if my phone rings more than four times now, the secretary (Edie) should answer (theoretically). Here's my new address and phone:

Bob Dietrich
MS 92-134
Tektronix, Inc. phone: (503) 645-6464 ext 1727
P.O. Box 580
Beaverton, Oregon 97007
U.S.A.

Of course, this list does not include the many more companies that supply Pascals for the xyz computer. Often (and why not?) these companies do a much better job than the companies that actually build the processors. You can draw your own conclusions from this list.

Validation Suite Reports

The University of Tasmania
Postal Address: Box 252C, G.P.O., Hobart, Tasmania, Australia 7001
Telephone: 23 0561. Cables 'Tauni' Telex: 58150 UNTAS

14th March, 1980

The Editor, Pascal News.

Validation Suite Report
This report to readers of Pascal News is intended to let everyone know of our intentions and plans. The demand for the validation package and response to it has almost swamped our capability of replying.

The current version 2.2 of the Validation Suite has been distributed to about 150 organizations or individuals, not counting the several thousands reached via Pascal News. As an indicator, the distribution list of our US distributor Rich Cichelli, is enclosed. Some suppliers are using the Validation Suite results in their advertising, and many are using it as a development tool. I have received a number of comparative reports, and have noticed a healthy competition to achieve 100% on the conformance/deviance tests.

We have almost completed an update to Version 2.3, which will correct the known errors in Version 2.2, and will include a few tests which were accidentally omitted in the first release. Unfortunately, even with the greatest care we could muster, several erroneous programs slipped through into the release of 2.2, and a few had features which caused them to fail on some processors for unrelated reasons. Version 2.3 is the response to such problems. However, it is still derived from the version of the Draft Standard printed in Pascal News and IEEE Computer, and known in ISO circles as ISO/TC97/SC5-N462.

American Microsystems
Basic Timesharing
Control Data Corporation
Data General
Digital Equipment Corporation
General Automation
Hewlett-Packard
Honeywell
IBM
Intel
Motorola
National Semiconductor
Texas Instruments
Three Rivers Computer
Varian division of Sperry Univac
Western Digital
Zilog
As soon as this is tested and released, we begin work on updating the whole package to the ISO Draft Standard now being circulated for voting. I estimate that this will take us about 2-3 months, for completely checking over 300 programs is non-trivial, and the insertions will require to be carefully drafted. The sources of change are primarily due to:

(i) areas in the earlier draft standards that were poorly drafted now being more precisely defined,
(ii) areas in the draft standard which have been altered, usually because N462 contained some mistake or ill-conceived change,
(iii) field experience with the package showing us weak spots in its attack strategies on compilers.

I should like to thank all those who have sent Brian, Rich or me copies of their results, or better still concise summaries and comments for the future. Your praise and criticisms help sustain us through a quite difficult piece of software engineering. Indeed we now realize that we should perhaps have written ourselves more tools at the start to carry through what I think to be a most significant piece of change in the software industry, and I am very much aware just how many contributions have gone up to make this effort.

May I simply continue to urge readers of Pascal News to keep on pushing the view that "correct is right" (with apologies to T.H. White), and to refuse to accept second-best.

Arthur Sale,
Professor of Information Science

PASCAL VALIDATION SUITE USERS

Oregon Software Inc.
Portland, Oregon 97201
Honeywell PMSC
Phoenix, Arizona 85029
Rational Data Systems Inc.
New York City, NY 10019
Advanced Computer Techniques
Arlington, Virginia 22209
Prime Computer
Framingham, Mass 01701
Hewlett Packard
Palo Alto, Calif 94304

Systems Engineering Labs
Ft. Lauderdale, Fla 33319
General Automation Inc.
Anaheim, Calif 92805
University of California at Santa Barbara
Santa Barbara, Calif 93106
Texas Instruments
Dallas, Texas 75222
National Semiconductor Corporation
Santa Clara, Calif 95051
Boeing Co.
Seattle, Washington 98124

Temek Corporation
Scottsdale, Arizona 85254
University of Waterloo
Waterloo, Ontario, Canada
Sperry Univac
Blue Bell, Pa. 19424
Perkin Elmer Corporation
Tinton Falls, NJ 07774
Boston Systems Office Inc.
Waltham, Mass 02154
Intel Corporation
Santa Clara, Calif 95051
General Research Corporation
Santa Barbara, Calif 93111
University of Minnesota
Minneapolis, Minn 55455
University of California at San Diego
La Jolla, Calif 92093
Intermetrics Inc.
Cambridge, Mass 02138
University of British Columbia
Vancouver, British Columbia, Canada
Virginia Polytechnical Institute & State University
Blacksburg, Va 24061

Digital Equipment Corporation
Tewksbury, Mass 01876
Phillips Laboratories
Briarcliff Manor, NY 10510
Honeywell NY12-3187
Minneapolis, Minn 55408
RCA-MSRD 127-302
Moorestown, NJ 08057
Boeing Co.
Seattle, Washington 98124
David Intersimone
Granada Hills, Calif 91344

Comshare Inc.
Ann Arbor, Michigan 48104
OCLC Inc.
Columbus, Ohio 43212
TRW C585
San Diego, Calif 92121
Medical Data Consultants
San Bernardino, Calif 92408
University of California at San Francisco
San Francisco, Calif 94143
Timeshare
Hanover, NH 03755
Fairchild Camera & Instrument Corp.
Mountainsview, Calif 94042
NCR Corporation
Copenhagen, Denmark
Process Computer Systems
Saline, Mich 48176
Vrije Universiteit
Amsterdam, The Netherlands
Scientific Computer Services
Glenview, Ill 60025
Burroughs Corporation
Goleta, Calif 93101

University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
Language Resources
Boulder, Colorado 80302
Jet Propulsion Lab
Pasadena, Calif 91103
Michigan State University
East Lansing, Mich 48824
Some time ago you acquired a version of the Pascal Validation Suite, either from us or from Rich Cichelli in the USA or from Brian Wichmann in the UK. If your version is up to date, you should have Version 2.2.

To briefly explain our numbering system for versions, the first digit identifies a major break in the evolution. Thus Version 1 related to the pre-1979 work derived from the Pascal User Manual and Report, and Version 2 is the completely revised package produced after receipt of the first public draft of a Pascal Standard (ISO/TC97/SC5 N462, known as Working Draft 3). The second number relates to a revision level within that major version.

With the release of Version 2.0, and its subsequent rapid evolution through 2.1 to 2.2, we have achieved a relatively stable product. It is by now quite well known that in the 350+ programs of the package there are a small set which are incorrect (they do not test what they ought to, or have a syntax error, or a convention error), and there is a small set which are not as well-designed as they might be (failing for reasons which are unrelated to their purpose).

Accordingly, while I was on sabbatical leave from the University of Tasmania in 1979/80, Brian Wichmann and staff at the National Physical Laboratories in England produced a new version 2.3 which attempts to correct these errors, and which adds a number of new tests together with old ones which were omitted from version 2 but were in version 1.

We will not distribute this version, and it will remain purely an internal revision level. Of necessity, the first production of a new level must be tested before release, and our testing of version 2.3 yields many issues which would have to be clarified before we could distribute it with the confidence in its quality that you are entitled to expect.

Even more cogently, we consider the revision of the validation package to conform to the new Draft Proposal (DP7185) to be even more important than tidying up the loose ends of an obsolete version level, and accordingly our efforts are now going into producing that version as soon as possible. It will be known as Version 3.0, and will take us at least two or three months to complete.

In this way we think we can avoid delays in the production of 3.0 and minimize the circulation of spurious tests and those which are relevant to N462 but not to DP7185 (or worse, reversed in the two versions...)

While undertaking the major revision required to produce the new version, we shall also attempt to simplify some aspects of testing. Since Version 3.0 will be a major revision, we shall issue it complete (i.e. not an update issue), but we intend in future to include a "last revision level" in the header of each test to facilitate identifying the latest changes.

Thank you for your support of our effort; we have over 150 subscribers now and the activity is certainly paying off in terms of quality of software and convenience to users. Best wishes for your future work.

Professor A.H.J. Sale

The University of Tasmania

Postal Address: Box 252C, G.P.O., Hobart, Tasmania, Australia 7001
Telephone: 23 0561. Cables 'Tasuni' Telex: 58150 UNTAS

11th March, 1980

Mr. P. Pickelmann,
Computing Centre,
University of Michigan,
1075 Beal Avenue,
Ann Arbor, Michigan
U.S.A. 48109

Dear Paul,

Thank you for your letter, which I have just read after returning to Tasmania from study leave in USA and Europe. I was very excited to read it, as it seems a very thorough piece of work, and just the sort of thing we hoped the package would do.

I have taken the liberty of sending a copy of your report to Pascal News for reprinting; if you want it kept private please write to Rick Shaw and say so, or send revisions. I have also sent a copy to the AAGC (Jeff Tobias) as he has told me that his field test version passes all conformance and practically all deviation tests! (or at least the correct tests)

I do not think that a tape with all three tests would be of great use to me at present as we are about to shift up one sub-level in the tests, and a new version level is three months away (to conform to the new Draft Standard). I think I can glean all I need from your very comprehensive report.

On your "Distribution problems", etc:
1. Charset: will investigate.
2. Printfiles: the distributed skeleton program will readily paginate; I will not put control characters in for the few installations that want them, at the expense of making 99% of installations strip them off. The printed version was printed by a slight modification of the skeleton.

Errors in test programs: will investigate; most have been reported frequently (sigh; complete correctness of 350+ programs too much for us; and flaws like 6.2.1-7 slip through.)

Specific suggestions

Clock would be less standard than process time. The name of a non-standard function is irrelevant; process time is deliberately chosen so as not to be in anyone's system (except ours) and to return results in standard metric units (seconds). Consequently inadvertent rubbish results are unlikely.

The suggestion about [1 mod bitsperword] illustrates only poor quality compilation techniques. Our compiler and the ICL 1900 one should realize that the result is in the range 0..(bitsperword-1) anyway. Consequently I would prefer to keep the algorithm transparent rather than introduce extraneous variables whose whole purpose is to optimize less-than-perfect implementations. (As a matter of interest, I have been musing over a version with very large sets here; our implementation will handle them too.)

6.3.1 § 6.4.5-5 are slips; our compiler has full significance, and all the others I used for testing had 10 or 12 or 16 characters up to release. We also forgot to run the full package with our STANDARD switch set to enable the compiler to report these.

6.8.5-4 Perhaps maxint is a bit severe? We are seeking implementations which allow 'virtual infinity' of case, to show quality. (Our compiler will handle maxint of course, but I wouldn't condemn a compiler that had a hash-table algorithm with packed one-word records and hence was limited to less than maxint values as the key.)

LOOP. Agree. Didn't realize that anyone was foolish enough to use loop-exit until talking with IBM implementors.

For-loops: you are tackling things which were left out of Version 2 because I could not resolve them in advance of the Draft Standard (or at least tried to influence the Standard first).

VERSION indication is a good idea, which we had already noted, but not in so clear a form. Thanks.

Finally, can you send me your size in shirts? We have a free gift to validators who do good work for Pascal...

Yours sincerely,

Arthur Sale, Information Science Department
The "Pascal Validation Suite" is a set of 318 Pascal programs designed to test a compiler for compliance with the draft Pascal standard. A full listing of the suite along with Arthur Sale's delightful introduction is in Pascal News, 16 (October 1979 arrived Jan. 80). The results of running the 3 Pascal compilers available on MTS are summarized below. A full report is in UNSP: PASCAL NEWS.

Version 2.2 of the suite was used. This corresponds to the version of the draft in Pascal News, 14 (Jan. 79). There are at least two newer drafts and a new version of the suite is coming.

If the number of tests failed seems disappointing, note that the designers took care to test those things which have changed from one definition of Pascal to the next, as well as those (mostly errors) which are hard to deal with.

<table>
<thead>
<tr>
<th>Test Type</th>
<th>#tests</th>
<th>Failed/Passed</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAFC</td>
<td>139</td>
<td>17/122</td>
</tr>
<tr>
<td>STBB</td>
<td>94</td>
<td>33/61</td>
</tr>
<tr>
<td>DEC</td>
<td>46</td>
<td>22/24</td>
</tr>
<tr>
<td>Quality</td>
<td>25</td>
<td>5/12</td>
</tr>
<tr>
<td>Extensions</td>
<td>1</td>
<td>1/0</td>
</tr>
<tr>
<td>Cost</td>
<td>$16.98</td>
<td>$10.20</td>
</tr>
</tbody>
</table>

The number of tests failed seems disappointing, note that the designers took care to test those things which have changed from one definition of Pascal to the next, as well as those (mostly errors) which are hard to deal with.

<table>
<thead>
<tr>
<th>Test Type</th>
<th>#tests</th>
<th>Failed/Passed</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAFC</td>
<td>139</td>
<td>17/122</td>
</tr>
<tr>
<td>STBB</td>
<td>94</td>
<td>33/61</td>
</tr>
<tr>
<td>DEC</td>
<td>46</td>
<td>22/24</td>
</tr>
<tr>
<td>Quality</td>
<td>25</td>
<td>5/12</td>
</tr>
<tr>
<td>Extensions</td>
<td>1</td>
<td>1/0</td>
</tr>
<tr>
<td>Cost</td>
<td>$16.98</td>
<td>$10.20</td>
</tr>
</tbody>
</table>

A Note on a Bit of Ambiguity

by: it is not

Parameter A parameter of any kind (value, var, procedure, or function) of a procedure or function.

Procedure Parameter A parameter of a procedure or function which is a procedure or function.
Conformance Tests

AAEC  STER  UBC

Number of tests passed = 122  113  118
Number of tests failed = 17  26  21

Failed Tests

AAEC
6.1.2-3,  6.1.8-3,  6.2.2-3,  6.3-1,  6.4.3.3-1,  6.4.3.3-4,
6.4.3.5-1,  6.4.3.5-2,  6.4.3.5-3,  6.5-1,  6.6.3.1-5,  6.6.3.4-2,
6.6.3.6-1,  6.6.3.9-7,  6.6.2-3,  6.6.4-4,  6.9-6
STER
6.1.6-2,  6.2.1-6,  6.2.2-3,  6.2.2-9,  6.4.2.2-2,  6.4.3-1,
6.4.3.3-10,  6.4.3.5-1,  6.4.3.5-3,  6.4.5-1,  6.5-1,  6.6.1-1,
6.6.3.1-5,  6.6.3.2-1,  6.6.3.3-1,  6.6.3.4-2,  6.6.5.2-3,  6.6.5-2,
6.6.5.2-5,  6.6.6.2-3,  6.6.6.4-1,  6.6.6.5-1,  6.7.2.4-3,  6.8.3.9-7,
6.9-4,  6.9-4 15
UBC
6.1.3-2,  6.2.2-3,  6.2.2-9,  6.4.3.5-1,  6.4.3.5-2,  6.4.3-5,  6.5-1,
6.5.3-4-1,  6.6.3.1-1,  6.6.3.1-3,  6.6.3.1-5,  6.6.3.4-2,
6.6.5.2-3,  6.6.5.2-5,  6.6.6.2-3,  6.7.2.5-2,  6.8.3.9-7,  6.9-4,
6.9-4 15

Details of failed tests:

AAEC
Only the first eight characters of identifiers and reserved words are used. Some longer identifiers look like reserved words.
Failed 6.1.2-3 and 6.3-1

UBC
Upper and lower letters are considered distinct in identifiers.
Failed 6.1.3-2

STER
Labels are compared as strings so leading zeros are significant.
Failed 6.1.6-2

AAEC
In "(*)" and "(*)" the starting and ending delimiters don't match but are considered the entire concept, which is what later
versions of the draft standard require.
Failed 6.1.8-2

STER
The program-parameters part of the program-heading is not optional.
Failed 6.2.1-6,  6.6.3.2-1,  6.6.3.3-1, and 6.6.5.5-1

AAEC, STER, UBC
When declaration for a type which is the domain of a pointer type appears after the declaration of the pointer type and there is a
more global type with the same name, the more global type is used for the domain of the pointer instead of the locally declared type.
Failed 6.2.2-3

STER, UBC
Assignment to a function identifier is not permitted from within
nest procedures and functions.
Failed 6.2.2-6.

STER
The cardinality of subranges must be less than Maxint. Programs
will run as long as these are never assigned a value greater than
min(subtype)*Maxint.
Failed 6.4.2.2-2 (error message, but runs)

STER
The tag-field is required in variant records.
Failed 6.4.3.3-1

AAEC
Empty record declarations containing a semicolon produce syntax
errors.
Failed 6.4.3.3-1

AAEC
The tag-field may not redefine an identifier elsewhere in the
declaration part.
Failed 6.4.3.3-4

STER
Case constants outside the tag-field subrange are not allowed,
which is what later versions of the draft standard require.
Failed 6.4.3.3-10

AAEC, STER, UBC
Pointers are not allowed within files.
Failed 6.4.3.5-1

AAEC
Null and length one lines have a blank appended when written.
Failed 6.4.3.5-2

STER, UBC
Null lines are replaced by length one lines when written.
Failed 6.4.3.5-2

STER
To solve the "interactive file problem" eof is undefined until
eof is checked.
Failed 6.4.3.5-2,  6.6.5.2-3
There is a bug where an eof check is need when it shouldn't be.
Failed 6.4.3.5-3

UBC
The end-of-line character is not not a
Failed 6.4.3.5-2

UBC
Local files (those other than program parameters) are not really
local. They must be provided by the user and all files with the
same name use the same file.
Failed 6.4.3.5-2,  6.4.3.5-3,  6.5.3-4-1,  6.6.3.1-3,  6.6.5.2-3
6.6.5.2-3,  6.9.4-15
AAEC
Peset does not do an implicit writeln (except with output)
Failed 6.4.3.5-3

STEP
Assignment to a var parameter whose type is an alias for the type
of the value assigned gives an error message and causes the
compiler to program interrupt.
Failed 6.4.5-1

AAEC, UBC
Records may not contain files.
Failed 6.5.1-1

STEP, UBC
An actual parameter of sce type for a var parameter which is a
subrange of that type is not allowed. This is what the draft
standard requires; the test is in error.
Failed 6.6.3.1-1

AAEC, STBB, UBC
Test has error. A parameter is included with a procedure parameter.
Failed 6.6.3.1-5

AAEC, STBB
The syntax for the par-list of procedure parameters is different.
UBC
Full specification(par-list) of procedure parameters is not allowed.
Failed 6.6.3.1-5, 6.6.3.4-2

AAEC, UBC
Cannot have procedure parameters with procedure parameters.
Failed 6.6.3.4-2

STEP, UBC
If the TMS-file which is used for a local file is not empty and
the first thing done is reset, the file is not empty and eof is
not true.
Failed 6.6.5.2-3

STEP
'of used with file being written causes an error.
Failed 6.6.5.2-5

STEP
Test 6.6.6.2-3 requires too much precision of real functions.
UBC
The expression Arctan(0)=0 yields false even though Arctan(0)
yields 0.
Failed 6.6.6.2-3

STEP
Ord returns different values when applied to variables of a
subtype and it's basetype which have the same value. Specifically
Ord(min(subtype))=0.
Failed 6.6.6.4-1

STEP
The expression "3 * (..)" causes a run error.
"ailed 6.7.2.4-3

UBC
The expression "(C,1.) \leq x" causes a run error.
Failed 6.7.2.5-2

AAEC
In a for loop the assignment is done before the second expression
is evaluated.
Failed 6.8.3.1-1

AAEC, STBB, UBC
Extreme value in for loops cause problems. UBC infinite loops,
AAEC and STBB cause run errors.
Failed 6.8.3.5-7

AAEC
Real numbers are converted differently at compile time than at run
time.
Failed 6.9.2-3.

AAEC, STBB, UBC
The formatting of reals when the field width given is too small
is wrong. Test is likely wrong, as the draft standard is not
clear. This section is changed in later drafts.
Failed 6.9.4-4

UBC
Strings are left justified, not right justified as they should be.
Failed 6.9.4-6

AAEC, UBC
'True ' instead of 'TRUE ' is used when writing booleans. This
may be changed in later versions of the standard.
Failed 6.9.4-7

STBB
Due to a bug, local files which are not global may not be used.
Release 3 will fix this and many other problems with files.
Failed 6.9.4-15
Deviance Tests

<table>
<thead>
<tr>
<th>Devisor of deviations detected</th>
<th>AAEC</th>
<th>STB</th>
<th>TEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failed 6.1.2-1, 6.1.7-7, 6.1.7-8, 6.1.7-11, 6.2.1-5, 6.2.2-4, 6.2.2-9, 6.3-6, 6.4.1-2, 6.4.1-3, 6.4.5-2, 6.4.5-3, 6.4.5-13, 6.4.5-15, 6.4.5-5, 6.6.2-5, 6.6.3-5-2, 6.6.3-6-2, 6.8.3-9-2, 6.8.3-9-3, 6.8.3-9-4, 6.8.3-9-5, 6.8.3-9-9, 6.8.3-9-10, 6.8.3-9-11, 6.8.3-9-13, 6.8.3-9-14, 6.8.3-9-16, 6.8.3-9-19</td>
<td>= 61</td>
<td>59</td>
<td>53</td>
</tr>
<tr>
<td>Failed 6.1.7-5, 6.1.7-6, 6.2.1-5, 6.2.2-4, 6.3-6, 6.4.1-2, 6.4.1-3, 6.4.5-2, 6.4.5-3, 6.4.5-13, 6.4.5-15, 6.4.5-5, 6.6.2-5, 6.6.3-5-2, 6.6.3-6-2, 6.8.3-9-2, 6.8.3-9-3, 6.8.3-9-4, 6.8.3-9-5, 6.8.3-9-9, 6.8.3-9-10, 6.8.3-9-11, 6.8.3-9-13, 6.8.3-9-14, 6.8.3-9-16, 6.8.3-9-19</td>
<td>= 32</td>
<td>31</td>
<td>38</td>
</tr>
</tbody>
</table>

Failed Tests

<table>
<thead>
<tr>
<th>Devisor of deviations detected</th>
<th>AAEC</th>
<th>STB</th>
<th>TEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failed 6.1.2-1, 6.1.7-7, 6.1.7-8, 6.1.7-11, 6.2.1-5, 6.2.2-4, 6.2.2-9, 6.3-6, 6.4.1-2, 6.4.1-3, 6.4.5-2, 6.4.5-3, 6.4.5-13, 6.4.5-15, 6.4.5-5, 6.6.2-5, 6.6.3-5-2, 6.6.3-6-2, 6.8.3-9-2, 6.8.3-9-3, 6.8.3-9-4, 6.8.3-9-5, 6.8.3-9-9, 6.8.3-9-10, 6.8.3-9-11, 6.8.3-9-13, 6.8.3-9-14, 6.8.3-9-16, 6.8.3-9-19</td>
<td>= 61</td>
<td>59</td>
<td>53</td>
</tr>
<tr>
<td>Failed 6.1.7-5, 6.1.7-6, 6.2.1-5, 6.2.2-4, 6.3-6, 6.4.1-2, 6.4.1-3, 6.4.5-2, 6.4.5-3, 6.4.5-13, 6.4.5-15, 6.4.5-5, 6.6.2-5, 6.6.3-5-2, 6.6.3-6-2, 6.8.3-9-2, 6.8.3-9-3, 6.8.3-9-4, 6.8.3-9-5, 6.8.3-9-9, 6.8.3-9-10, 6.8.3-9-11, 6.8.3-9-13, 6.8.3-9-14, 6.8.3-9-16, 6.8.3-9-19</td>
<td>= 32</td>
<td>31</td>
<td>38</td>
</tr>
</tbody>
</table>

Undetected Extensions

<table>
<thead>
<tr>
<th>AEC</th>
<th>STB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failed 6.1.5-6, 6.6.3-5-12, 6.6.4-9, 6.9.4-12</td>
<td></td>
</tr>
<tr>
<td>Failed 6.1.5-6, 6.9.4-9, 6.9.4-12</td>
<td></td>
</tr>
</tbody>
</table>

Details of deviations not detected

<table>
<thead>
<tr>
<th>AEC</th>
<th>STB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failed 6.1.2-1</td>
<td></td>
</tr>
<tr>
<td>Failed 6.1.7-5</td>
<td></td>
</tr>
</tbody>
</table>

AEC Strings are comparable with arrays of length n, not just those with index 1..n.
Failed 6.1.7-6, 6.4.3.2-5

STB Strings are comparable with arrays of subrange of char.
Failed 6.1.7-7 and 6.1.7-8

URC Null strings are accepted.
Failed 6.1.7-11

AEC,STB,UHC Declared but unused labels are allowed.
Failed 6.2.1-5

AEC,STB,UHC With in a scope a global name may be used then redefined.
Failed 6.2.2-4

AEC Function identifiers may be assigned to outside the bounds (text) of the function.
Failed 6.2.2-9

STB,URC "*" (but not "-") may be used on things of type CHAR, string, and scalars, not just integers and reals.
Failed 6.3-2, 6.3-3, 6.3-4, 6.3-5, and 6.7.2.2-9

AEC A name may be used in it's own definition e.g. "const ten=ten;"
Failed 6.3-2, and 6.4.1-2

AEC,UHC A global name may be used within a record which redefines that name.
Failed 6.4.1-3

URC Allows packed anything not just (direct) structures.
Failed 6.4.1-1, and 6.4.3.1-2

STB Pointers to undeclared types may be used, but not dereferenced.
Failed 6.4.4-2

URC Comparisons are allowed between different types.
Failed 6.4.5-10 and 6.4.5-11

AEC,STB,UHC The "Pn" definition of type equivalence rather than the stricter current definition.
Failed 6.4.5-3, 6.4.5-4(AEC,STB), 6.4.5-5, 6.4.5-13

AEC A comparable type is allowed as a var parameter.
Failed 6.4.5-2
STB
Missing POPASC procedures go undetected.
Failed 6.6.1-6

AAEC,STBR,UBC
Missing assignment to a function identifier goes undetected.
Failed 6.6.2-2

AAEC
Actual function parameters returning types compatible with the formal function parameter are allowed.
Failed 6.6.3.5-2

AAEC,UBC
Actual function parameters returning types compatible with the formal function parameter are allowed.
Failed 6.6.3.5-2

AAEC,UBC
Actual and formal procedure parameters may have parameters which are compatible, not just the same.
Failed 6.6.3.6-2, and 6.6.3.6-3

STBR
Trunc and Round with integer arguments get by.
 Failed 6.6.6.3-4

AAEC,STBR,UBC
Goto's are allowed between then and else parts of if statements and between cases in a case statement. A later draft allowed this, but it looks like it's out of the current one, which is too bad at least in the case of the case statements.
Failed 6.8.2.4-2, and 6.8.2.4-3

AAEC,STBR,UBC
Goto's are allowed into structured statements. See the test for some interesting implications of this and the definition in the draft.
Failed 6.8.2.4-8

STBR
Real case selectors get by (when the case constants are reals).
Failed 6.8.3.5-10

UBC
Components of records are allowed as for loop variables.
Failed 6.8.3.9-11

AAEC,STBR,UBC
Non-local variables are allowed as for loop variables.

Assignments to for loop variables inside the loop are allowed.

Nested for loops with the same variable are allowed. In STBR this doesn't cause infinite loops, since at the top of the loop the variable gets the value it would have if not changed.
Failed 6.8.3.9-2, 6.8.3.9-3, 6.8.3.9-9, 6.8.3.9-14, 6.8.3.9-16, and 6.9.3.9-19

STBR,UBC
Output may be used even if it doesn't appear in the program header.
Failed 6.10-1

STBR,UBC
Write may be used without specifying a file even when output

has been declared.
Failed 6.10-3

Details of extensions not detected

STBR,UBC
'e' for 'E' is allowed in real constants. Later drafts allow this.
Failed 6.1.5-6

STBR
Subranges in case lists are not flagged as extensions. (Version 2S of the compiler doesn't allow them though).
Failed 6.8.3.9-12

AAEC,STBR,UBC
Zero and negative field widths are allowed. Later drafts may allow this.
Failed 6.9.4-9,

STBR,UBC
Write works with unpacked arrays of char, not just packed ones.
Failed 6.9.4-12

Tests failed for non-conformance

UBC
Fully specified parameter lists are not allowed.
Failed 6.6.3.5-2, 6.6.3.6-2, 6.6.3.6-3, 6.6.3.6-4, and 6.6.3.6-5

AAEC
Procedure parameters may have only value parameters.
Failed 6.6.3.6-3, and 6.6.3.6-4

AAEC,UBC
Locp is a reserved word.
Failed 6.8.3.9-5, 6.8.3.9-13, and 6.8.3.9-14
Error Handling

<table>
<thead>
<tr>
<th>Number of errors detected</th>
<th>23</th>
<th>24</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of errors not detected</td>
<td>23</td>
<td>22</td>
<td>24</td>
</tr>
</tbody>
</table>

Failed Tests

AAC

6.2.1-7, 6.4.1-3-5, 6.4.3-3-6, 6.4.3-7, 6.4.3-9-6, 6.4.3-12, 6.4.6-7, 6.4.6-8, 6.6.2-6, 6.6.5-2-6, 6.6.5-2-7, 6.6.5-3-3, 6.6.5-3-4, 6.6.5-3-5, 6.6.5-3-6, 6.7.2.2-6, 6.7.2.4-1, 6.8.3.5-4, 6.8.3.5-9, 6.8.3.9-5, 6.8.3.9-17

STER

6.2.1-7, 6.4.3-3-5, 6.4.3-3-6, 6.4.3-7, 6.4.3-8, 6.8.6-7, 6.4.6-8, 6.6.2-6, 6.6.5-2-6, 6.6.5-2-7, 6.6.5-3-3, 6.6.5-3-4, 6.6.5-3-5, 6.6.5-3-6, 6.7.2.2-6, 6.7.2.4-1, 6.8.3.9-5, 6.8.3.9-17

UBC

6.2.1-7, 6.4.3-3-5, 6.4.3-3-6, 6.4.3-7, 6.4.3-8, 6.6.6-2-6, 6.6.6-5-2-6, 6.6.6-5-2-7, 6.6.6-5-3-3, 6.6.6-5-3-4, 6.6.6-5-3-5, 6.6.6-5-3-6, 6.6.6-5-3-9, 6.7.2.2-6, 6.7.2.4-1, 6.8.3.9-5, 6.8.3.9-6, 6.8.3.9-17

Details of Failed Tests:

AAC, STER, UBC

Use of undefined variables is not detected.
Failed 6.2.1-7, 6.4.3-3-5, 6.4.3-3-6, 6.4.3-3-8, 6.6.2-6, 6.8.3.9-5, 6.8.3.9-6

AAC

Use of an null record causes an operation exception.
STER

Use of a null record is considered an incompatible assignment.
UBC

Use of a null record which is therefore an undefined variable is not detected.
Failed 6.4.3-3-12

AAC, STER, UBC

Variant errors are undetected
Failed 6.4.3-3-5

AAC, STER, UBC

Set assignments out of range are not detected. Comments in 6.7.2.4-1 say something about "operations on overlapping sets" but I can't find section 6.7.2.4.
Failed 6.4.6-8(AAC, STER), 6.4.6-8(AAC, STER), 6.7.2.4-1

STER

Get with eof true is not detected.
Failed 6.6.5.2-2
Quality Measures

<table>
<thead>
<tr>
<th></th>
<th>AAEc</th>
<th>STrB</th>
<th>UEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tests run</td>
<td>18</td>
<td>23</td>
<td>18</td>
</tr>
<tr>
<td>Number incorrectly handled</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Failed Tests

<table>
<thead>
<tr>
<th></th>
<th>AAEc</th>
<th>STrB</th>
<th>UEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.2-1, 6.1.3-3, 6.1.8-4, 6.4.3.4-5, 6.6.1-7, 6.8.3.5-2, 6.8.3.9-18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.1.8-4, 6.4.3.2-4, 6.8.3.5-2, 6.8.3.5-6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.1.9-4, 6.4.3.2-4, 6.8.3.5-2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tests not run

<table>
<thead>
<tr>
<th></th>
<th>AAEc</th>
<th>STrB</th>
<th>UEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.6.6.2-6, 6.6.6.2-7, 6.6.6.2-8, 6.6.6.2-9, 6.6.6.2-10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Details of failed tests:

<table>
<thead>
<tr>
<th></th>
<th>AAEc</th>
<th>STrB</th>
<th>UEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>No warning is given for long identifiers, and only the first eight characters are used.</td>
<td>Failed 5.2.2-1, 6.1.3-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No warning is given for a (short) comment with a missing &quot;}&quot;.</td>
<td>Failed 6.1.8-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Array(integer) confuses the compiler and causes an obscure things at run-time.</td>
<td>Failed 6.4.3.2-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1 mod bitsetword) is not done correctly. Worked when changed to [t] where t was 0..bitsminus1.</td>
<td>Failed 6.4.3.4-5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Procedure nesting is limited to 6 levels (main,P1..P5).</td>
<td>Failed 6.6.1-7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No warning is given for an impossible case, one whose label is outside the subrange of the selector. This maybe an error in later drafts.</td>
<td>Failed 6.8.3.5-2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Implementation-Assurance

<table>
<thead>
<tr>
<th></th>
<th>AAEc</th>
<th>STrB</th>
<th>UEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tests run</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Number incorrectly handled</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Tests Incorrectly Handled

<table>
<thead>
<tr>
<th></th>
<th>AAEc</th>
<th>STrB</th>
<th>UEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>There was an integer overflow evaluation trunc((a+b)-a) which should have returned 16. Failed 6.6.6.2-11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set of char should work, but doesn't always Failed 6.4.3.4-2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test Results

<table>
<thead>
<tr>
<th></th>
<th>AAEc, STrB, UEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maxint = 2,147,483,647</td>
<td></td>
</tr>
<tr>
<td>Set of char should work, but the test fails.</td>
<td></td>
</tr>
<tr>
<td>Sets of 0..1000 are allowed. Range is 0..2047.</td>
<td></td>
</tr>
<tr>
<td>Sets of 0..1000 are allowed. Any subrange with 2049 or fewer members can be the base type for a set. Set constructor works only on scalars and subranges, not integers.</td>
<td></td>
</tr>
<tr>
<td>Sets of 0..1000 not allowed. Base types may have up to 256 members. Set constructor only works with numbers in 0..255.</td>
<td></td>
</tr>
<tr>
<td>There is an integer overflow in trunc(expr=16.0), only with this program (??).</td>
<td></td>
</tr>
<tr>
<td>Beta=15, T=6, End=0, Ngcd=1, Machexp=-5, Negexp=-6, Iexp=7, Emaxexp=65, maxexp=63, eps=9.5367e+37, epsneg=5.96046e-03,</td>
<td></td>
</tr>
<tr>
<td>xmax=5.397653e-72,xmax=7.2371555e+75</td>
<td></td>
</tr>
<tr>
<td>Beta=15, T=16, End=0, Ngcd=1, Machexp=-13, Negexp=-19, Iexp=7, Emaxexp=65, maxexp=63, eps=2.22044605e-16, epsneg=1.38777788e-17,</td>
<td></td>
</tr>
<tr>
<td>xmax=5.397653e-72,xmax=7.2371555e+75</td>
<td></td>
</tr>
<tr>
<td>Tests 6.7.2.3-2, 6.7.2.3-3</td>
<td></td>
</tr>
<tr>
<td>Booleans expressions are fully evaluated. UBC has option to use</td>
<td></td>
</tr>
</tbody>
</table>
partial evaluation.

STEP

McCarthy evaluation of boolean expressions is used.

Tests 6.8.2.2-1, 6.8.2.2-2
AAFC, UBC
Tests show selection before evaluation.

STEP

First test shows selection before evaluation, second evaluation before selection.

Tests 6.9.4.5, 6.5.4-11
AAFC
Default field widths for integers 12, reals 24, booleans 5.
Exponents have 2 digits.

STEP

Default field widths for integers 12, reals 24, booleans 5.
Exponents have 2 digits.

UBC
Default field widths for integers 10, reals 22, booleans 10.
Exponents have 2 digits.

Test 6.9.4.6-1
AAFC, UBC
These tests used upper case identifiers declared in lower case and had 'e' in real constants.

6.6.6.2-6, 6.6.6.2-7, 6.6.6.2-8, 6.6.6.2-9, 6.6.6.2-10

Results of Tests

Test 6.1.3-3
AAFC
Only the first 8 characters of an identifier are used.

STER, UBC
Tests report more than 20 characters of identifiers used. STBR uses all characters, UBC uses 32.

Test 6.4.3.3-9
AAFC, STER, UBC
The tag-field in records is not checked. Test reports 'exact correlation'.

Test 6.4.3.4-5
Measures the time for Warshall's algorithm on an 80x80 matrix.

Original uses array(0..79.) of array(0..4) of set of 0..15.
Modified uses array(0..79.) of set of 0..79.

<table>
<thead>
<tr>
<th>Time (sec)</th>
<th>Size (words/bits)</th>
<th>Time (sec)</th>
<th>Size (words/bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAFC</td>
<td>0.087</td>
<td>502/16064</td>
<td>0.021</td>
</tr>
<tr>
<td>STER</td>
<td>0.065</td>
<td>400/12890</td>
<td>0.020</td>
</tr>
<tr>
<td>UBC</td>
<td>0.099</td>
<td>670/21440</td>
<td>0.033</td>
</tr>
</tbody>
</table>

Test 6.7.2.2-4
AAFC, STER, UBC
Div and mod with negative operands are as in the latest draft. Div B = Trunc(A/B), and mod returns the remainder of div, that is it has the same sign as the quotient.

Test 6.8.3.9-16
AAFC
After a for loop the loop variable may have a value which is out of range.

STER, UBC
After a for loop the loop variable has value of the final expression.

Test *** (All)
The total cost of running all 318 programs was:

AAFC $16.98
STER $10.20 done Compile and Execute, several compilations per run
UBC $38.75 done with LCAUXC
Burroughs B6700

PASCAL VALIDATION SUITE REPORT

Pascal Processor Identification

Computer: Burroughs B6700
Processor: B6700 Pascal version 3.0.001
(University of Tasmania compiler)

Test Conditions

Tester: R.A. Freak (implementation/maintenance team member)
Date: March 1980
Validation Suite Version: 2.2

Conformance Tests

Number of tests passed: 137
Number of tests failed: 1

Details of failed tests:
Test 6.4.3.5-1 fails because a file of pointers or a file of sets is not permitted.

Deviance Test

Number of deviations correctly detected: 83
Number of tests showing true extensions: 2 (2 actual extensions)
Number of tests not detecting erroneous deviations: 9 (5 basic causes)
Number of tests failed: 0

Details of extensions:
Test 6.1.5-6 shows that the lower case e may be used in real numbers (for example 1.602e-20). This feature has been included in the new draft standard.
Test 6.10-1 shows that the file parameters in the program heading are ignored in B6700 Pascal.

Details of deviations not detected:
Test 6.1.2-1 shows that nil may be redefined.
Tests 6.2.2-4, 6.3-6 and 6.4.1-3 show that a common scope error was not detected by the compiler.
Tests 6.8.2.4-2, 6.8.2.4-3 and 6.4.2.4-4 show that a goto between branches of a statement is permitted.
Test 6.9.4-9 shows that integers may be written with a negative format.
Test 6.10-3 shows that the file output may be redefined at the program level.

Error Handling

Number of errors correctly detected: 33
Number of errors not detected: 13 (4 basic causes)

Details of errors not detected: The errors not detected fall into a number of categories -
Tests 6.4.3.3-5, 6.4.3.3-6, 6.4.3.3-7 and 6.4.3.3-8 indicate that no checking is performed on the tag field of variant records.
Tests 6.6.5.2-1 and 6.6.5.2-7 indicate that a file buffer variable can be altered illegally and a put may be performed on an input file.
Tests 6.6.5.3-3, 6.6.5.3-4, 6.6.5.3-5 and 6.6.5.3-6 fail because dispose always returns a nil pointer in B6700 Pascal and no check is performed on the pointer parameter.
Tests 6.6.5.3-7, 6.6.5.3-8 and 6.6.5.3-9 fail because no checks are inserted to check pointers after they have been assigned a value using the variant form of new.

Implementationdefined

Number of tests run: 15
Number of tests incorrectly handled: 0
Details of implementation-dependence:
Test 6.4.2.2-7 shows maxint to be 549755813887.
Tests 6.4.3.4-2 and 6.4.3.4-4 show that large sets are allowed. The maximum set size is 65536 elements. A set of char is permitted.
Test 6.6.6.1-1 shows that some standard functions can be passed as parameters. Those which use in-line code cannot be passed as parameters.
Test 6.6.6.2-11 details some machine characteristics regarding number formats.
Tests 6.7.2.3-2 and 6.7.2.3-3 show that boolean expressions are fully evaluated.
Tests 6.8.2.2-1 and 6.8.2.2-2 show that a variable is selected before the expression is evaluated in an assignment statement.
Tests 6.9.4-5 and 6.9.4-11 show that the default size for an exponent field on output is 2; for a real number it is 15; for a boolean 5 and the size varies for integers according to the value being written.
Test 6.10-2 indicates that a rewrite on the standard file output is permissible.
Tests 6.11-1, 6.11-2 and 6.11-3 show that the alternative comment delimiters have been implemented, as have the alternative pointer symbols. No other equivalent symbols have been implemented.

Quality Measurement
Number of tests run: 23
Number of tests incorrectly handled: 0

Results of tests:
Test 5.2.2-1 shows that identifiers are distinguished over their whole length.
Test 6.1.3-3 shows that more than 20 significant characters may appear in an identifier, in fact, the number of characters in a line is allowed.
A warning is produced if a semicolon is detected in a comment (test 6.1.8-4).

Tests 6.2.1-8, 6.2.1-9 and 6.5.1-2 indicate that large lists of declarations may be made in each block.
An array with an integer indextype is not permitted (test 6.4.3.2-4).
Test 6.4.3.3-9 shows that variant fields of a record occupy the same space, using the declared order.
Test 6.4.3.4-5 (Warshall's algorithm) took 0.698304 secs CPU on the Burroughs B6700 and 158 bytes.
Tests 6.6.1-7, 6.8.3.9-20 and 6.8.3.10-7 show that procedures, for statements and with statements may each be nested to a depth greater than 15.
Tests 6.6.6.2-6, 6.6.6.2-7, 6.6.6.2-8, 6.6.6.2-9 and 6.6.6.2-10, tested the sqrt, atan, exp, sin/cos and ln functions and all tests were successfully completed, without any significant errors in the values.
Test 6.7.2.2-4 shows that div has been implemented consistently for negative operands, returning trunc. mod returns for the remainder of div.
Test 6.8.3.5-2 shows that case constants must be of the same type as the case-index, if the case-index is a subrange, and a warning is given for case constants which cannot be reached.
Test 6.8.3.5-8 shows that a large case statement (256 selections) is permissible.
Test 6.8.3.9-18 indicates that range checking is always used in a case statement after a for statement to check the for variable.
Test 6.9.4-10 shows that file buffers are flushed at the end of a block and test 6.9.3-14 indicates that recursive I/O using the same file is allowed.

Extensions
Number of tests run: 1
Test 6.8.3.5-14 shows that the otherwise clause in a case statement has been implemented according to the accepted convention.
Data General Eclipse

PASCAL VALIDATION SUITE REPORT

PASCAL Processor Identification

Computer: Data General Eclipse S/130
Processor: Medical Data Consultants BLAISE (PASCAL P4 v4 DEC 1979)

Test Conditions

Tester: Ted C. Park
Date: April, 1980
Validation Suite Version: 2.2

General Comments

1. The overall quality and completeness of the validation programs is excellent.
2. The orthogonality of the programs is poor. Oftentimes many things are checked in one test. For instance, my compiler supports TRUNC but not ROUND. Since these are checked in the same test, this causes problems.
3. The skeleton program seems like a good idea but in actual practice it did me very little good. I wonder if it's really helpful to anyone else.
4. The skeleton program requires a "dummy" terminating program at the end of the validation suite. There is none.
5. The first line of program 6.8.3.4-1 is missing a comma.
6. Program 6.6.1-6 is missing a semicolon on the next to the last statement.

The PASCAL-P4 Subset

MDC "BLAISE" is based on PASCAL-P4 which is a known subset of PASCAL as described in Jensen and Wirth. It was not clear at the outset how a subset compiler would react to the validation programs. All the programs were submitted to the compiler and although many were invalid due to the known design restrictions, I am pleased to report that the compiler either accepted each program or printed appropriate diagnostic messages in every case. No program caused any system failure or crash either at compile or run time.

The known design constraints of PASCAL-P4 (See PASCAL NEWS #11, Page 70) are listed below.

NIL is a predeclared constant
FORWARD is a reserved word
Only the alternate form of comment delimiters are allowed
No WAVINT
No TEXT
No ROUND
No PAGE
No DISPOSE
No RWRITE
No RESET
No PACK
No UNPACK
The program heading is not required
Every variant record must have a tag field
No user declared files or associated features (BLAISE does not support GET or PUT)
No output of BOOLEANs
No output of REALS in fixed notation
No formal parameter functions or procedures
No subrange set constructors
64 character ASCII character set which implies upper case letters only.
No literal text strings longer than 16 characters.
8 character limit on identifier lengths.

Since the upper case only and 16 character literal string length restrictions applied universally to almost all programs, they were all adjusted accordingly. Other than that, no changes were made to any of the programs. The results are reported below.

Conformance Tests

Number of tests attempted: 139
Number of tests invalid due to known design restrictions: 31
Number of tests passed: 102
Number of tests failed: 6
Details of Failed Tests

Test 6.1.5-2 failed because long REALs are not accepted by the compiler, however, a warning message was issued.

Test 6.2.2-3 failed due to a scoping error.

Test 6.4.3.5-4 failed because no end of line was inserted at final buffer flush.

Test 6.8.2.4-1 failed because non-local GOTOs are not allowed.

Test 6.8.3.5-4 failed because of the large table generated for a sparse CASE statement.

Test 6.8.3.9-1 failed because the index of a FOR statement was set up before the final expression of the FOR statement was evaluated.

Deviance Tests

Number of tests attempted: 94
Number of tests invalid due to known design restrictions: 21
Number of tests passed: 50
Number of tests failed: 23

Details of Failed Tests

Test 6.1.7-8 failed because any character may be assigned to an element whose type is subrange of CHAR.

Test 6.2.2-4 fails to detect the scope overlap.

Test 6.3-5 fails because it allows a signed character constant.

Test 6.3-6 fails because it allows a constant to be used in its own declaration.

Test 6.4.1-3 fails because it allows a type to be used in its own declaration.

Test 6.4.5-2 fails because subranges of the same host are treated as identical.

Test 6.4.5-3 fails because similar arrays are treated as identical.

Test 6.4.5-4 fails because similar records are treated as identical.

Test 6.4.5-5 fails because similar pointers are treated as identical.

Test 6.6.2-5 fails because assignment to the function identifier is not required.

6.6.4.6-6 fails because SDCC and PRED are allowed for REALs.

Test 6.7.2.2-9 fails because the unary plus is allowed for a variable of type CHAR.

Test 6.8.2.4-2 fails because jumps between branches of an IF statement are allowed.

Test 6.8.2.4-3 fails because jumps between branches of a CASE statement are allowed.

Test 6.8.3.9-2 fails because assignment to the FOR index is allowed.

Test 6.8.3.9-3 fails because assignment to the FOR index is allowed.

Test 6.8.3.9-4 fails because assignment to the FOR index is allowed.

Test 6.8.3.9-9 fails because a non-local variable is allowed as a FOR index.

Test 6.8.3.9-14 fails because a global variable is allowed as a FOR index.

Test 6.8.3.9-16 fails because the FOR index can be read.

Test 6.8.3.9-19 fails because nested FORs with the same index are not detected.

Test 6.9.4-6 fails because zero and negative field widths allowed are for integer output.

Test 6.9.4-12 fails because output of non-packed arrays is allowed.

Error Handling Tests

Total tests attempted: 46
Number of tests invalid due to known design restrictions: 13
Number of tests passed: 8

* Number of tests passed only if "DEBUG" option selected: 11
Number of tests failed: 14

Details of Failed Tests

Test 6.2.1-7 local values are not undefined prior to definition.
Test 6.4.3.3-5 other variants do not cease to exist when tag field changed.
Test 6.4.3.3-6 variants are not undefined prior to definition.
Test 6.4.3.3-12 empty field is not flagged as undefined prior to definition.

* Test 6.4.6-4 out of range not detected on integer assignment.
* Test 6.4.6-5 out of range not detected on integer parameter passing.
* Test 6.4.6-6 out of range not detected on integer array index.
* Test 6.4.6-7 out of range not detected on set assignment.
* Test 6.4.6-8 out of range not detected on set parameter passing.
* Test 6.5.3.2-1 out of range not detected on two dimensional integer array index.
* Test 6.5.4-1 pointer equals NIL not detected at use.
* Test 6.5.4-2 pointer undefined not detected at use.
Test 6.6.2-6 function having no value assigned to it as undetected.
Test 6.6.5.3-7 assignment compatibility of records not checked.
Test 6.6.5.3-8 assignment compatibility of records not checked.
Test 6.6.5.3-9 assignment compatibility of records not checked.

* Test 6.6.6.4-4 SUCC function applied to last value not detected.
* Test 6.6.6.4-5 PRED function applied to first value not detected.
* Test 6.7.2.2-3 divide by zero not detected.
Test 6.7.2.2-8 mod by zero not detected.
* Test 6.7.2.4-1 out of range SET values not detected.

Test 6.8.3.9-5 undefined FOR indexed after loop not detected.
Test 6.8.3.9-6 undefined FOR index after zero pass loop not detected.
Test 6.8.3.9-17 nested FOR using same index not detected.

Implementation-Defined Tests

Test 6.4.2.2-7 no MAXINT
Test 6.4.3.4-2 SET of CHAR allowed
Test 6.4.3.4-4 SET base-type size 0...63
Test 6.6.6.1-1 functions not allowed as parameters
Test 6.6.6.2-11 all floating-point tests OK
Test 6.7.2.3-2 (A AND B) fully evaluated
Test 6.7.2.3-3 (A OR B) fully evaluated
Test 6.8.2.2-1 left side of array assignment evaluated before right side
Test 6.8.2.2-2 left side of pointer assignment evaluated before right side
Test 6.8.4-5 two digits written for exponent
Test 6.9.4-11 IFM=10 BFN=20 BFN not allowed
Test 6.10-2 rewrite not allowed
Test 6.11-1 [] not allowed for comments
Test 6.11-2 equivalent symbols for ^ ; ; ; = [ ] not allowed
Test 6.11-3 equivalent symbols for < > <= >= <> not allowed

Quality Tests

Test 6.2.2-1 identifiers not distinguished past 8 characters
Test 6.1.3-3 identifier significance is 8 characters
### Test 6.1.8-4
No help in locating unclosed comment.

### Test 6.2.1-8
50 types allowed.

### Test 6.2.1-9
50 labels allowed.

### Test 6.4.3.2-4
Integer not allowed as index type.

### Test 6.5.1-2
Long declaration lists allowed.

### Test 6.6.1-7
Procedures may be nested only 10 deep.

### Test 6.6.2-6
SQRT is OK.

### Test 6.6.2-7
ARCTAN is OK.

### Test 6.6.2-8
EXP is OK.

### Test 6.6.2-9
SIN and COS are OK.

### Test 6.6.2-10
LN is OK.

### Test 6.7.2.2-4
DIV is OK — MOD returns remainder.

### Test 6.8.3.5-2
 Impossible branch of CASE not detected.

### Test 6.8.3.5-8
256 CASES allowed.

### Test 6.8.3.9-18
FOR index is just bumped along without checking.

### Test 6.8.3.9-20
15 nested FORs allowed.

### Test 6.8.3.10-7
15 nested WITHs allowed.

### Test 6.8.4-10
Output is not flushed at end of job.

### Test 6.9.4-14
Recursive I/O allowed.

### Extension Tests

### Test 6.8.3.14
'OTHERWISE' extension not implemented.

---

**DEC VAX 11/780**

---

**VAX 11 Pascal Validation Report**

**Pascal Processor Identification**

- **Computer:** VAX 11/780
- **Processor:** VAX 11 Pascal V1.0-1

**Test Conditions**

- **Time:** 1980 01 21
- **Test runs carried out by S. Matwin and B. Silverman**
- **Test annotation and analysis by S. Matwin**
- **Validation Suite version:** 2.2

**Conformance Tests**

- **Number of tests passed:** 127
- **Number of tests failed:** 12, 8 basic causes

**Details of failed tests:**

- Test 6.4.3.3-1 shows that empty record is not implemented.
- Test 6.4.3.3-4 shows that the processor does not allow tag field redefinition.
- Tests 6.4.3.5-1 and 6.5.1-1 show that the function EXP does not pass accuracy test.
- Test 6.8.3.5-4 shows that CASE label range is limited to 10000.
- Test 6.8.3.9-7 shows that MAXINT is too big as an extreme value in a for statement, leads to overflow.
- Test 6.8.4.4-3, 6.9.4-4, 6.9.4-7, and 6.9.5-1 fail with a component of a packed structure as an actual variable parameter. This will happen in any compiler, written in Pascal, as the parameters of READ will be variable. On the other hand the Standard prohibits 'the use of components of variables of any packed type as actual variable parameters'.
- Test 6.9.4-15 shows that WRITE without the file parameter refers to a locally defined file.

**Deviance Tests**

- **Number of deviations correctly detected:** 67
- **Number of tests not detecting erroneous deviations:** 24
  (6 basic causes)

**Details of deviations not detected:**

- Test 6.1.2-1 shows that the reserved word nil may be redefined.
- Test 6.1.5-6 shows that the processor allows small letter 'e' as an exponent indicator (which is sometimes claimed to be an extension).
- Tests 6.2.2-4 and 6.3-6 show that in some circumstances the processor does not detect the use of an identifier prior to its definition.
Tests 6.4.5-2 thru 6.4.5-5 and 6.4.5-13 show that the processor requires the compatibility of the types of formal and actual parameters, rather than type identity.

Test 6.6.2-5 shows that the processor does not check the occurrence of at least one assignment to the function name in the function block.

Tests 6.8.2.4-2 thru 6.8.2.4-4 show that the processor allows jumps between branches of an if and a case statement.

Tests 6.8.3.9-2 thru 6.8.3.9-5, 6.8.3.9-13 thru 6.8.3.9-16 and 6.8.3.9-19 show that the processor permits some restrictions imposed on a for statement. The processor prohibits neither the assignment to the control variable nor the use of that variable after the completion of the loop. Other deviations of that class are:
- control variable can be a formal parameter or a global variable
- reading into a control variable is allowed
- non-local control variable combined with recursion leads to an infinitely looping program.

**Error Handling**

Number of errors correctly detected: 13
Number of errors not detected: 31

**Details of errors not detected**

Tests 6.2.1-7 and 6.4.3.3-12 show that the undefined values are not detected by the processor.

Tests 6.4.3.3-5 thru 6.4.3.3-8 show that the existence of a particular variant in a record variable is not tested by the processor.

Tests 6.4.6-4 thru 6.4.6-8, 6.5.3.2-1 and 6.7.2.4-1 show that the processor tests only the static compatibility, without checking the appropriateness of the actual value during run-time (unlike, e.g., Zurich Pascal-2 compiler).

Test 6.6.2.5-6 show that there is no dynamic checking of the fact whether the name is assigned to the function name.

Tests 6.6.2.5-6 and 6.6.5.2-7 show that the parameter called by value can be changed inside the procedure in case of a buffer variable.

Tests 6.6.5-3 and 6.6.5-4 show that the procedure DISPOSE does not check correctness of its parameter.

Tests 6.6.5-3 and 6.6.5-6 show that both an actual variable parameter and an element of a record-variable-list of a with statement can be referred to by a pointer parameter of DISPOSE.

Tests 6.6.5-3-7 thru 6.6.5.3-9 show that the restrictions on the variable, created by the second form of NEW, are not implemented.

Tests 6.6.6-4 and 6.6.6-5 that SUC and PRED can produce values from beyond the enumeration type.

Test 6.6.6.4-7 shows that the function OR does not check the correctness of its parameter.

Tests 6.8.3.5-5 and 6.8.3.6-6 show that there is no dynamic checking of the value of the case selector.

Test 6.8.3.9-17 shows that two nested for statements can use the same control variable.

**Implementation defined**

Number of tests run: 16
Number of tests incorrectly handled: 1

**Details of the implementation-dependencies**

Test 6.4.2.2-7 shows LARGE to be 2147483647

Tests 6.4.3.4-2 and 6.4.3.4-4 show that INTEGER is allowed, that the negative elements in a set are not allowed, and that elements must not exceed 255.

Tests 6.6.1-1 fails because formal functions are implemented following the Revised Report rather than the Standard.

Tests 6.7.2.3-2 and 6.7.2.3-3 show that Boolean expressions are fully evaluated.

Tests 6.8.2.2-1 and 6.8.2.2-2 show that selection precedes evaluation in the binding order.

Tests 6.9.4-5 and 6.9.4-11 show that the default fields are:
- 10 for integer
- 16 for Boolean
- 16 for real.

Test 6.10-2 shows that REWRITE on the standard file OUTPUT is possible.

Tests 6.11-1 thru 6.11-3 show that only alternate comment delimiters (and no other equivalent symbols) are permitted.

**Quality Measurement**

Number of tests run: 23
Number of tests incorrectly handled: 1

**Details of results**

Tests 5.2.2-1 and 6.1.3-3 show that there is no other limit on the length of the identifiers than the length of the line, although only the first 15 characters are significant.

Test 6.18-4 shows that in case of an unclosed comment the text is swallowed without any diagnostics.

Tests 6.1.2-9 and 6.1.2-3 show that large type- and label-lists are allowed.

Test 6.4.3.2-4 shows that INTEGER is not allowed as an index type.

Test 6.4.3.3-9 shows that fields in a record are stored in the order of their appearance in the field list.

Test 6.4.3.4-5 (Warshall's algorithm) took 129 milliseconds of CPU time.

Tests 6.6.6.2-6 thru 6.6.6.2-10 were completed with some errors, requiring separate analysis.

Test 6.7.2.2-4 shows that div and mod have been implemented consistently for negative operands: quotient = trunc(a/b), mod returns remainder of div.

Test 6.8.3.5-2 shows that 'impossible' paths through case statements are not signalled by the processor.

Test 6.8.3.5-8 shows that a large number of case labels is allowed.

Test 6.8.3.9-18 shows that the value of the control variable after the completion of a for loop is in the range of its type (and is equal...
Tests 6.8.3-20 and 6.8.3.10-7 show that for and with statements can be nested to a depth exceeding 15.

Test 6.9.4-10 shows that flushing the buffer of the output file occurs at the end of the program.

Test 6.9.4-14 shows that recursive I/O using the same file is not possible.

Extensions

Number of tests run: 1

Test 6.8.3-14 shows that otherwise clause is implemented, although one statement (rather than a sequence of them) is permitted between otherwise and end.
program file; the termination logic has to be altered slightly anyway.
- The "END." for test 6.6.1-7 does not begin in column 1.

Conformance Tests

Number of tests passed: 113
Number of invalid tests: 3
Number of tests failed: 22 (14 causes)
Number of irrelevent tests: 3
Number of tests detecting bugs in compiler: 6

Invalid tests

6.4.3.5-1 PTRTOI, meant as a type, declared as a variable.
6.6.3.1-1 contains an actual VAR parameter non-identical in type to the formal parameter. The compiler passed this test when the error was corrected.
6.9.4-7 TRUE is written in a field of 5; when read back, the program expects it to be written left justified, in contrast to the standard which says that values should be written right justified.
Irrelevant tests

6.1.3-6, 4.2.2-6 Compiler uses upper case only.
6.6.5-1 not a test program.
Tests detecting bugs in compiler

6.2.2-3 When typing a pointer to a type NODE, the compiler uses an definition of NODE from an outer block rather than a new definition of NODE appearing later on in the same block.
6.4.3-3 causes a bad instruction to be generated.
6.4.5-1 produces an irrelevant error message relating to file assignment.
6.5.2-3 blew up on a RESET to an un-initialized internal file using Release 3.1. The test passes using Release 3.2.
6.7.2-4-1 blew up on the expression A * [ ] = [ ].

Details of Failed Tests

6.1.6-2 Label in compared for equality as strings rather than integers and thus labels "6" and "0006" are considered distinct.
6.2.1-6, 6.6.3.2-1, 6.6.3.3-1 Compiler expects at least one parameter in the program heading.
6.2.2-8 Compiler does not allow assignment to the value of a function within an inner block of that function.
6.4.2-2 The maximum cardinality of a subrange is restricted to the value of MAXINT; compiler gives a warning and runs correctly, but only because the subrange is subsequently treated as equivalent to type INTEGER.
6.4.3.1-1 Untagged variants are not permitted.
6.4.3.1-10 Case constants outside the tag field subrange are not allowed.
6.4.3.5-2, 6.9.3-1-1 Implementation uses fixed length records, even for text files; an empty line thus results in a record of blanks, rather than a single line-marker character.

6.6.3.4-1 A different syntax is used for declaring the parameter types of formal procedure/function parameters - only the types of the parameters are expected.
6.6.6.2-3, which tests the real-valued standard arithmetic functions, failed on the accuracy tests for EXP and SQRT.
6.6.6.4-1 Compiler computes ORD(x) with respect to the declared subrange to which x belongs, rather than with respect to the underlying base type.
6.8.1-9-7 When using values near MAXINT in a FOR loop, compiler gave an INTEGER OVERFLOW run error.
6.9.4-4 The second width specifier for formatting reals is not implemented.
6.9.4-6 The width specifier for strings must be a constant in the current implementation.

Deviance

Number of tests passed: 54
Number of tests showing deviance: 34 (17 causes)
Number of tests failed: 5
Number of tests detecting bugs: 3

Details of tests showing deviance

6.1.7-5, 6.9.4-12 because PACKED and UNPACKED structures are treated as equivalent; i.e., the compiler makes no distinction between the two even for storage requirements.
6.1.7-6, 6.4.3.2-5 Strings are compatible with all arrays of CHAR provided the lengths match.
6.2.1-5 If an identifier is declared as a label no error is produced if it is not subsequently referenced in a GOTO.
6.2.2-4 Use of a type identifier is permitted according to its definition in an outer block despite its redefinition in an inner block.
6.3-2, 3, 4, 5, 6.7.2-9 shows signed constants of inappropriate types (e.g. strings) are allowed.
6.4.3-11, which tries to assign a value to an empty field in a record, blows up during semantic analysis (PASS 2 of the compiler).
6.4.5-3 and 6.4.5-13, which is identical), 6.4.5-4, 5 fail because the compiler uses structural equivalence rather than name equivalence of types.
6.4.4-2 The compiler fails to flag references to a pointer variable that points to a record type that is never defined.
6.6.1-6 Shows that compiler does not catch the lack of a subsequent full declaration for a procedure declared to be FUNCTION (the program is allowed to run, even though that routine is actually called); this is a bug. This test, as supplied, contained a missing semicolon.
6.6.2-5 Compiler does not detect the lack of an assignment of a value to a function within the function block.
6.6.6.4 Integer arguments to TRUNC and ROUND are permitted. (Such arguments are coerced to real as they would be in any other instance where reals are expected).
6.8.2.4-2,3,4 show the compiler allows jumps into IF and ELSE parts, and into CASE branches.

6.8.3.5-10 Compiler allows REAL CASE labels with a corresponding REAL CASE selector; test executes correctly.

6.8.3.9-2,3,4,14,16, 6.8.3.9-9,19 Show that there are practically no restrictions on FOR loop control variables; they can be assigned to or read in within (or outside) the loop body, and declared in any block. However, altering control variables do not affect the number of loop iterations; an altered value is retained only throughout the iteration in which it is changed, since the compiler uses a hidden temporary variable as the true control variable.

6.8.3.9-2,3,4,14,16, 6.8.3.9-9,19 Show that there are practically no restrictions on FOR loop control variables: they can be assigned to or read in within (or outside) the loop body, and declared in any block. However, altering control variables do not affect the number of loop iterations; an altered value is retained only throughout the iteration in which it is changed, since the compiler uses a hidden temporary variable as the true control variable.

6.9.4-9 Shows that there are practically no restrictions on FOR loop control variables: they can be reassigned to or read in within (or outside) the loop body, and declared in any block. However, altering control variables does not affect the number of loop iterations; an altered value is retained only throughout the iteration in which it is changed, since the compiler uses a hidden temporary variable as the true control variable.

6.9.4-9 Shows that there are practically no restrictions on FOR loop control variables: they can be reassigned to or read in within (or outside) the loop body, and declared in any block. However, altering control variables does not affect the number of loop iterations; an altered value is retained only throughout the iteration in which it is changed, since the compiler uses a hidden temporary variable as the true control variable.

6.9.4-9 Shows that there are practically no restrictions on FOR loop control variables: they can be reassigned to or read in within (or outside) the loop body, and declared in any block. However, altering control variables does not affect the number of loop iterations; an altered value is retained only throughout the iteration in which it is changed, since the compiler uses a hidden temporary variable as the true control variable.

6.10-1 OUTPUT is not required to be listed in the program heading when output is directed to that file in the program.

6.10-3 Shows OUTPUT can be redefined as a variable within the program block.

6.8.3.5-12 shows compiler allows ranges as case labels.

Tests showing bugs in compiler 6.4.3.3-11, 6.4.4-2, 6.6.1-6 (described above)

Tests showing extensions 6.8.3.5-12,13, 6.8.3.9-10 show ranges are allowed as case labels, and that this extension is implemented safely.

Tests failed 6.6.3.5-2, 6.6.3.6-2,3,4,5 all failed because the compiler expects a different syntax for declaring the parameter types of formal procedure/function parameters.

Comments on passed tests 6.1.5-4 Decimal point not followed by a digit in a real number flagged as an error, but the program is allowed to run because no ambiguity is present in the case tested by the program.

6.1.7-11 A null string is flagged, but the program is allowed to run with a blank substituted.

6.1.8-5 Nested comments are permitted if the alternate delimiter symbols are used.

6.9.4-8 When real format is used to output an integer, no error is given but the program is allowed to run.

Error handling tests

Number of tests passed: 25
Number of tests failed: 23
Number of invalid tests: 1

Details of failed tests 6.2.1-7 No error message is given when an undefined variable is used.

6.4.1-3-5,6 show no run-time check on tag values is performed when referencing variants.

6.4.3.3-7,8 failed because the compiler does not allow untagged variants.

6.4.3.3-7,8 failed because the compiler does not allow untagged variants.

6.6.2-6 Shows no check is made whether a function receives a value.

6.6.5.3-2 No EOF error given. This test fails because the implementation uses fixed length records for text files, and thus short lines are padded with blanks.

6.6.5.3-6,7 No error is given if a file component variable is an actual parameter to a procedure that does I/O to the file and thus alters the file component.

6.6.5.3-3,4 fail because DISPOSE is not implemented; no check is made on the validity of its arguments. Similarly, 6.6.5.3-6 shows no error is given when a pointer used in selection of a WITH control variable is disposed.

6.6.5.3-5 would fail if the test program were valid; the parameter A should be a VAR parameter.

6.6.5.3-7,8 show that no error is given if a variable returned by NEW containing tagged variants is used in its entirety.

6.6.5.3-5,6 When the value of a case selector <> any of the labels, no error message is given.

6.8.3.9-5,6,17 show that a FOR loop control variable is accessible outside the loop. After normal execution of the loop, it has the final value of the range. No error is given for nested FOR loops using the same control variable; the program iterates the expected number of times.

Implementation defined tests

Number of tests run: 15
Number of tests detecting bugs: 1

Details of Implementation dependence 6.4.2.2-7 shows MAXINT = 2147483647.

6.4.3.4-2 shows sets of CHAR are allowed.

6.4.3.4-4 shows the maximum set cardinality > 1000.

6.6.6.2-11 ran to completion, but some inconsistencies occurred (i.e., XMIN <> BETA**MINEXP).

6.7.2.3-2,3 show short circuit evaluation of expressions is performed.

6.8.2.2-1 shows selection is performed before evaluation in
A[I] := SIDEFFECT(I). By contrast, test 6.8.2.2-2 shows
6.9.4-5 shows 2 digit exponents in output of real numbers.
6.9.4-11 detected a bug in RELEASES 3.0, 3.1. It shows the default field widths to be:
   integer: 12
   boolean: 14
   real: 9
in contrast to the User manual and earlier releases, in which these formats are 12, 6, 14, respectively. This bug has been repaired in RELEASE 3.2.

6.10-2 shows REWRITE(OUTPUT) is not allowed.

6.11-1 shows the alternate comment convention is allowed; the delimiters must be pairwise matched, thus allowing code sections to be commented out.

6.11-2,3 show equivalent symbols %, =, GT, LT, GE, LE, NE, are not allowed. @ is used instead of the EBCDIC translation of up-arrow.

Quality tests

   Number of tests run: 22
   Number of tests detecting bugs in compiler: 6
   Number of tests not performed: 1

5.2.2-1, 6.1.3-3 show identifiers are distinguished over their whole length, but the compiler gives no indication the programs do not conform (i.e., contain identifiers with > 8 character significance). The compiler permits identifiers of up to 256 characters.

6.1.8-4 shows compiler gives no indication of unclosed comments.

6.2.1-8,9, 6.6.1-7, 6.8.3.9-20, 6.8.3.10-7 show a large number of label and type declarations, deeply nested (>15 levels) procedures, FOR loops, and WITH statements are permitted. However, test 6.8.3.5-8, which contains a heavily populated CASE statement, caused a compile time data structure to overflow at case 152.

6.7.2.2-4 shows DIV and MOD are implemented consistently, and that MOD yields the remainder of DIV.

6.9.4-10 shows that the output buffer is flushed at the end of the program.

6.8.3.5-2 shows the compiler does not detect that a case label, while contained in the underlying type, lies outside the subrange to which the selector belongs.

6.4.3.3.9 shows the ordering of the representation of variant fields is the same as the order of declaration.

6.6.6.2-6,7,8,9,10, which test the standard real-valued arithmetic functions, gave a mean relative error between E-06 and E-07 in the interval tests. The special argument tests gave fairly good results. Most identity tests gave zero, as required; those that did not were within E-06 relative to the arguments.

6.8.3.9-18 shows the value of a FOR statement control variable after normal termination of the loop is the specified upper limit.

6.9.4-14 shows "recursive" I/O is allowed.

Test not performed

6.4.3.4-5 could not be run because timing is currently not implemented in the CMS version.

Tests demonstrating compiler bugs

6.4.3.2-4 shows compiler accepts an array with an index type of INTEGER, but the resulting program does not run correctly.

6.6.6.2-6,7,8,9,10 all crashed at run-time using Release 3.1. The bug has been fixed in Release 3.2.

Extensions

Number of tests run: 1

Test 6.8.3.5-14 did not compile; the compiler supports the OTHERWISE extension to the CASE statement but OTHERWISE <statement> replaces END rather than preceding it as in the proposed standard extension.
PASCAL VALIDATION SUITE REPORT

Authored by:
I.E. Johnson, E.N. Miya, S.K. Skedzieleweski

Pascal Processor Identification
Computer: Univac 1100/81
Processor: University of Wisconsin Pascal version 1.0 release A

Test Conditions
Testers: I.E. Johnson, E.N. Miya.
Date: April 1980
Validation Suite Version: 2.2

General Introduction to the UW Implementation

The UW Pascal compiler has been developed by Prof. Charles N. Fischer. The first work was done using the P4 compiler from Trondheim, then the NOSC Pascal compiler written by Mike Ball was used, and now all development is done using the UW Pascal compiler.

There are two UW Pascal compilers: one produces relocatable code and has external compilation features, while the other is a "load-and-go" compiler, which is cheaper for small programs. Most tests were run on the "load-and-go" version. Both compilers are 1-pass and do local, but not global optimization. The UW compiler is tenacious and will try to execute a program containing compile-time errors. This causes problems when running the Validation Suite, since programs that are designed to fail at compile time will appear to have executed.

Conformance Tests

Number of Tests Passed: 123
Number of Tests Failed: 16

Details of Failed Tests
Test 6.4.3.5-1 failed on the declaration of an external file of pointers (only internal files of pointers are permitted).
Tests 6.4.3.5-2, 6.4.3.5-3 and 6.9.1-1 failed due to an operating system "feature" which returns extra blanks at the end of a line. This problem affects EOLN detection.
Test 6.5.1-1 failed because the implementation prohibits files that contain files.
Tests 6.6.3.1-5 and 6.6.3.4-2 failed because the current version of this implementation prohibits passing standard functions and procedures as parameters.
Test 6.6.5.3-1 failed to assign an already locked tag field in a variant record, but the standard disallows such an assignment (Error in test?)
Test 6.6.5.4-1 failed to pack because of a subscript out of range. MACC notified.
Test 6.6.6.2-3 failed a nine-digit exp comparison. Univac uses 8 digit floating point.
Test 6.6.6.5-2 failed test of ODD function (error with negative numbers).
Test 6.8.2.4-1 failed because non-local GOTO statements are not allowed by this implementation.
Test 6.8.3.4-1 failed to compile the "dangling else" statement, giving an erroneous syntax error.
Tests 6.9.4-1 and 6.9.4-4 failed do unrecoverable I/O error. Problem referred to MACC.
Test 6.9.4-7 failed to write boolean correctly. UW right-justifies each boolean in its field; the proposed ISO standard requires left-justification.

Extensions

Number of Tests Run: 1

Details of Tests
Test 6.8.3.5-14 shows that an OTHERWISE clause has been implemented in the case statement.

Deviance Tests

Number of Deviations Correctly Handled: 77
Number of Deviations Incorrectly Handled: 14
Number of Tests Showing True Extensions: 2

Details of Extensions
Test 6.1.5-6 shows that a lower case e may be used in real numbers.

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under NASA Contract NAS7-100.
Test 6.1.7-11 shows that a null string is accepted by this implementation.

Details of Incorrect Deviations

Tests 6.2.2-4, 6.3-6, 6.4.1-3 show errors in name scope. Global values of constants are used even though a local definition follows; this should cause a compile-time error.

Tests 6.4.5-3, 6.4.5-5 and 6.4.5-13 show that the implementation considers types that resolve to the same type to be "equivalent" and can be passed interchangeably to a procedure.

Test 6.6.2-5 shows a function declaration without an assignment to the function identifier.

Test 6.8.3.9-4 the for-loop control variable can be modified by a procedure called within the loop. No error found by implementation.

Tests 6.8.3.9-9, 6.8.3.9-13 and 6.8.3.9-14 show that a non-local variable can be used as a for-loop control variable.

Test 6.9.4-9 shows that a negative field width parameter in a write statement is accepted. It is mapped to zero.

Test 6.10-1 shows that the implementation substitutes the default file OUTPUT in the program header. No error message.

Test 6.10-4 shows that the implementation substitutes the existence of the program statement. We know that the compiler searched first but found source text (error correction).

Tests 6.1.8-5 and 6.6.3.1-4 appear to execute; this occurred after the error corrector made the obvious changes.

Error Handling

Number of Errors Correctly Detected: 29

Number of Error Not Detected: 17

Details of Errors Not Detected

Tests 6.2.1-7, 6.4.3.3-6, 6.4.3.3-7, 6.4.3.3-8 and 6.4.3.3-12 show that the use of an uninitialized variable is not detected. Variant record fields are not invalidated when the tag changes. 6.4.3.3-12 incorrectly printed "PASS" when it should have printed "ERROR NOT DETECTED".

Test 6.6.2-6 shows the implementation does not detect that a function identifier has not been assigned a value within the function. The function should be undefined. The quality of the test could be improved by writing the value of CIRCLERADIUS.

Test 6.6.5.2-2 again runs into the EOLN problem.

Test 6.6.5.2-6 shows that the implementation fails to detect the change in value of a buffer variable when used as a global variable while its dereferenced value is passed as a value parameter. This should not cause an error, and none was flagged. However, when the char was changed to a var parameter no error was detected, either.

Test 6.6.5.2-7 shows that the implementation fails to detect the change in a file pointer while the file pointer is in use in a with statement. This is noted in the implementation notes.

Test 6.6.5.3-5 shows the implementation failed to detect a dispose error; but again, the parameter was passed by value, not by reference! (Error in test)

Tests 6.6.5.3-7 and 6.6.5.3-9 show that the implementation failed to detect an error in the use of a pointer variable that was allocated with explicit tag values.

Tests 6.6.6.3-2 and 6.6.6.3-3 show that trunc or round of some real values. 2**36 does not cause a run time error or warning. In those cases, the value returned was negative. Error reported to MACC.

Tests 6.7.2.2-6 and 6.7.2.2-7 show that the implementation failed to detect integer overflow.

Tests 6.8.3.9-5 and 6.8.3.9-6 show that the implementation does not invalidate the value of a for-loop control variable after the execution of the for-loop. Value of the variable is equal to the last value in the loop. These tests could be improved by writing the value of m.

Implementation Defined

Number of Tests Run: 15

Number of Tests Incorrectly Handled: 0

Details of Implementation Definitions

Test 6.4.2.2-7 shows maxint equals 34359738367 (2**35-1).

Test 6.4.3.4-2 shows that a set of char is allowed.
Test 6.4.3.4-4 shows that 144 elements are allowed in a set, and that all ordinals must be $\geq 0$ and $\leq 143$.

Test 6.6.1-1 shows that neither declared nor standard functions and procedures (nor Assembler routines) be passed as parameters.

Test 6.6.2-11 details a number of machine characteristics such as:

\[
\begin{align*}
XMIN &= \text{Smallest Positive Floating Point #} = 1.4693679E-39 \\
XMAX &= \text{Largest Positive Floating Point #} = 1.7014118E+38
\end{align*}
\]

Tests 6.7.2.3-2 and 6.7.2.3-3 show that boolean expressions are fully evaluated.

Tests 6.8.2.2-1 and 6.8.2.2-2 show that expressions are evaluated before variable selection in assignment statements.

Test 6.9.4-5 shows that the output format for the exponent part of a real number is 2 digits. Test 6.9.4-11 shows that the implementation defined default values are:

- Integers: 12 characters
- Boolean: 12 characters
- Reals: 12 characters

Test 6.10-2 shows that a rewrite to the standard file output is not permitted.

Tests 6.11-1, 6.11-2, and 6.11-3 show that the alternative comment delimiter symbols have been implemented; all other alternative symbols and notations have not been implemented. In addition, it is interesting that the compiler's error correction correctly substituted "[*" for "(*" and ":=" for ":=" as well as a number of faulty substitutions.

**Quality Measurement**

**Number of Tests Runs:** 23

**Number of Tests Incorrectly Handled:** 2

**Results of Tests**

Test 5.2.2-1 shows that the implementation was unable to distinguish very long identifiers (27 characters). Test 6.1.3-3 shows that the implementation uses up to 20 characters in distinguishing identifiers.

Test 6.1.8-4 shows that the implementation can detect the presence of possible unclosed comments (with a warning). Statements enclosed by such comments are not compiled.

Tests 6.2.1-8, 6.2.1-9, and 6.5.1-2 show that large lists of declarations may be made in a block (Types, labels, and var).

Test 6.4.3.2-4 attempts to declare an array index range of "integer". The declaration seems to be accepted, but when the array is accessed (A[I][maxint]), an internal error occurs.

Test 6.4.3.3-9 shows that the variant fields of a record occupy the same space, using the declared order.

Test 6.4.3.4-5 (Warshall's algorithm) took 0.1356 seconds CPU time and 730 unpacked (36-bit) words on a Univac 1100/81.

Test 6.6.1-7 shows that procedures may not be nested to a depth greater than 7 due to implementation restriction. An anomalous error message occurred when the fifteenth procedure declaration was encountered; the message "logical end of program reached before physical end" was issued at that time, but a message at the end of the program said "parse stack overflow".

Tests 6.6.6.2-6, 6.6.6.2-7, 6.6.6.2-8, 6.6.6.2-9, and 6.6.6.2-10 tested the sqrt, atan, exp, sin/cos, and ln functions. All tests ran, however, typical implementation answers (which use the Univac standard assembler routines) were slightly smaller than Suite computed. Error typically occurred around the 8th digit (Univac floating-point precision limit).

Test 6.7.2.2-4 The inscrutable message "inconsistent division into negative operands" appears. We think it means that $I \mod 2$ is NOT equal to $I \div 2$. Problem reported to MACC.

Test 6.8.3.5-2 shows that case constants must be in the same range as the case-index.

Test 6.8.3.5-8 shows that a very large case statement is not permissible ($\geq 256$ selections). A semantic stack overflow occurred after 109 labels.

Test 6.8.3.5-18 shows the undefined state is the previous state at the end of the for-loop. The range is checked.

Test 6.8.3.9-20 shows for-loops may be nested to a depth of 6.

Test 6.8.3.10-7 shows while-loops may be nested to a depth of 7.

Test 6.9.4-10 shows that the output buffer is flushed at the end of a program.
Test 6.9.4-14 shows that recursive I/O is permitted using the same file.

Concluding Comments

The general breakdown of errors is as follows:

I/O
These problems are intimately tied to the EXEC 1100 operating system and its penchant to pad blanks on the end of a line. There is no plan to try to correct this problem. Does an external file of pointers make sense?

Changes in the standard
Jensen and Wirth (second edition) was used as the standard for development of this compiler. Since there are discrepancies between it and the ISO proposed standard, several deviations occurred. The compiler will be brought into conformance on most of these errors when some standard is adopted.

Restrictions
Some restrictions will be kept, even after a standard is adopted. GOTO's out of procedures will probably never be implemented, but STOP and ABORT statements have been added to the language to alleviate the problem.

Bugs
Several previously unknown bugs were found by running the validation suite. Professor Fischer has been notified, and corrections should be included in the next release of the compilers.

One area that should be emphasized is the clarity of the diagnostics produced by the compiler. All diagnostics are self-explanatory, even to the extent of saying "NOT YOUR FAULT" when an internal compiler error is detected. A complete scalar walk-back is produced whenever a fatal error occurs. The compiler attempts error correction and generally does a very good job of getting the program into execution.

The relocatable compiler has extensive external compilation features. A program compiled using these facilities receives the same compile-time diagnostics as if it were compiled in one piece.

Machine-dependent Implementations

Burroughs B6700/7700 (Tasmania)

Dear Bob,

Here is the latest information on the Pascal implementation for the Burroughs B6700/7700 series, as developed at the University of Tasmania. It still exists, and has been distributed quite widely. A new manual has just been produced which sets new standards of excellence for us, and is available presumably to subscribers who have paid our annual fee (to cover postage, etc).

We have been working on the compiler to make it conform to the draft Standard (a moving target at present), and I believe the current version includes the procedural parameter feature now that this seems to have stabilized. It is pleasing to note that our attitude towards checks is paying off, as shown when we recently uncovered three different usages in the P4 compiler where undefined values of variables were tested against well-defined values. No doubt these bugs are now widely distributed through the Pascal community!

Enquiries should not be addressed to me here (where I am on leave), but rather to Pascal Support, Dept of Information Science, University of Tasmania, Box 252C GPO, Hobart, Tasmania 7001. Don't forget the airmail stamp.

Best wishes,

Arthur Dale

---

Professor: H.B. Griffith, S.A. Robinson (Pure Mathematics); P.T. Landsberg (Applied Mathematics); J.W. Craggs (Engineering Mathematics); D.W. Bevan (Computer Studies); T.M.F. Smith (Statistics).
Mr. R. Shaw,
Digital Equipment Corp.,
5775 Peachtree-Dunwoody Road,
Atlanta, Georgia.
U.S.A.

Dear Rick,

I have recently updated the B6700/B7700 Pascal compiler to level 3.0.001. This compiler conforms to the Working Draft Standard, as published in Pascal News #14, fairly well. A copy of the updated Pascal Validation Suite Report concerning this compiler is enclosed.

We are in the process of distributing this compiler to all those installations which are currently using our Pascal system. The distribution should be complete by the time you receive/publish this letter.

We are also producing an updated Pascal Reference Manual to reflect the new compiler. The manual has just come to the printers and we will distribute copies to users of our Pascal System when it returns. Allow a month or so.

Enclosed is an updated checklist describing the new compiler.

Yours sincerely,

Roy A. Freak
Information Science Department
The compiler has been stable in code for some time, reflecting its basic integrity. However, new features are added from time to time, and notified to users as patches or as a new version release. The department accepts FTR notices and will attempt to fix those which warrant such attention. Some modifications have taken place as a result of user feedback. The compiler was especially designed not to generate dangerous code to the MCP, and no system crashes have been attributed to it since the first few months of testing, 3 years ago, and then only three.

7. STANDARD

The compiler conforms fairly well to the Pascal Standard as published in Pascal News #14. We intend to update the compiler when a Pascal standard is accepted by ISO. The compiler performs better than most during testing by the Pascal Validation Suite. Briefly, the following restrictions and extensions apply:

Restrictions: Program heading; reserved word `program` is synonymous with `procedure`; file parameters are ignored after program heading.


File attributes in declaration. Format declarations and record oriented I/O available. Extensive Burroughs-compatible compiler options (Pascal control comment option mode not implemented). Ability to link in externally compiled subprograms.

8. MEASUREMENT

Compiles about 20% slower than Fortran or Algol, but in about 2/3 their space (for test programs about 4-5K words on average instead of 8-10K). Elapsed compilation times similar, though Pascal slower. Speed should be improved by eventual tuning.

Executes at the same speed as Fortran and Algol (code is similar and optimal) and takes generally longer elapsed resident time primarily due to MCP intervention to create new segments for record structures (not present in Fortran/Algol). Elapsed resident time about 20% greater than equivalent Algol.

9. RELIABILITY

Excellent. Since the early testing three years ago, no system crashes have been attributed to Pascal. The compiler is now in use at 28 sites throughout the world. It has been in use since 76/10 at University of Tasmania. First released to outside sites in 77/4.

10. DEVELOPMENT METHOD

Compiler which generates 36700/B7700 code files which are directly executed by the B6700 MCP. Written in 36700 Algol with two intrinsics written in Espol. Hand-coded using Pascal-P as a guide/model. All other paths offered much more difficulty due to special nature of machine/system. Person-month details not kept, but project proceeds in fits and starts as teaching and other activities intervene. Project has been undertaken largely by two people: Professor A.H.J. Sale and R.A. Freak with some support from T.S. McDermott.

11. LIBRARY SUPPORT

With release 3.0.001 of the Pascal compiler, the system has the ability to link in externally compiled subprograms written in another language. There is no facility available for separately compiling Pascal subprograms (not standard) so the only method of binding involves a Pascal host and a subprogram written in another language. The system contains an extended set of pre-defined mathematical functions.
The new distributor for Pascal-6000 for East Asia and Australia is now:

Pascal Coordinator
University Computing Centre: H8B
University of Sydney
Sydney, N.S.W. 2006, Australia
Phone: 61-02-292 3491

Tony Gerber is finishing his studies and passed the responsibilities on to Brian Roussel.

DEC LSI-11 (SofTech)

The UCSD version of Pascal is available from SofTech for $350 (includes operating system, compiler, editor, etc.). A FORTRAN that compiles to P-code is also available. For more information and processors that are supported, contact:

SofTech Microsystems
4944 Black Mountain Road
San Diego, California 92126

1. All right, title, and interest in VAX-11 Pascal/Unix are the property of Digital Equipment Corporation (DEC).
2. Requestors for VAX-11 Pascal/Unix must have a license for the VMS version of VAX-11 Pascal from DEC. An object code license is required for the VAX-11 Pascal/Unix object code, a source code license for the VAX-11 Pascal/Unix source code.
3. The VAX-11 Pascal/Unix system will be distributed for a copy charge of US $50.00, payable to the University of Washington. Distribution will be on magnetic tape provided by UW. Please send your request, together with a check or purchase order, to:
   Department of Computer Science
   University of Washington
   Mail Stop FR-35
   Seattle, WA 98195

Further information can be obtained by contacting
   Professor Hellmut Golde (206) 543-9264

4. Requestors must sign the sublicense agreement attached to this announcement and return it to UW with the order. Please use the proper site identification so that the VMS license can be verified.
5. UW welcomes comments, suggestions and bug reports from users. Although no regular maintenance will be provided by either DEC or UW, a best effort will be made by UW to correct bugs for subsequent releases of VAX-11 Pascal/Unix. Any updated versions will require an additional copy fee.

The VAX-11 Pascal/Unix compiler does not implement all features of VAX-11 Pascal. However, the VAX-11 Pascal manuals available from DEC are sufficient to use VAX-11 Pascal/Unix. The following features are not currently supported by VAX-11 Pascal/Unix:

1. Value initialization.
2. %Include directive.
3. Calls to VMS library routines and system services. However, calls to the C library and Unix services are available.
4. The VMS debugger, and hence the DEBUG option. However, users may use the Unix absolute interactive debugger, adb(I).
5. The library functions/procedures DATE, TIME, and CLOCK.
6. Standard functions/procedures as procedure parameters.
In addition, a few restrictions are imposed under VAX-11 Pascal/Unix, as follows:

1. Since procedure linking is done by the Unix loader, all procedure names on nesting level 1 (main program level) and all external procedure names must differ in their first 7 characters. These names should not contain the character '§'.

2. The command language interface is different to conform with Unix.

3. Only standard Unix sequential files are supported. Hence the OPEN statement is limited to the form

   OPEN('<file variable>', '<unix file name>', '<file history>')</p>

   The specifications of <record access mode>, <record type>, and <carriage control> are ignored. Also, FORTRAN type carriage control is not available. The VMX procedure FIND has not been implemented.

Beyond these restrictions, every effort has been made to make the two compilers compatible. There are some minor differences in expressions using library procedures and in input-output conversions, due to different algorithms.

Hewlett Packard now distributes a version of Pascal for their HP 1000 system. For details, contact a sales office.

IBM Series/1 (Massey U.)

Pascal has been implemented at Massey University, Palmerston North, New Zealand for the IBM Series/1.

Hardware Requirements

Ability to support a 64K byte user partition using the R.P.S. operating system.

Major Restrictions

1. Files may not be declared. Four standard files are made available. These may be used as input or output files or (non-standardly) as direct I/O files.

2. Some standard functions are not implemented - in particular the mathematical functions SIN, COS etc. However, selected functions may easily be implemented if required.

3. Limited to 16 bit sets, although some built in routines to handle 48 bit sets are available.

Structure

The compiler is based on the P4 portable Pascal compiler written by:

Authors: Urs Ammann, Kesav Nori and Christian Jacobi

Address: Institut fuer Informatik
Eidg. Technische Hochschule
Ch-8096 Zuerich.

It runs in two passes, (production of the P4 code and conversion of the P4 code to Series/1 code), and employs several storage overlays (not overlays as implemented in R.P.S.). All of the compiler, except the special environment (small assembler program) in which it runs, is written in Pascal. It can compile the main body of the first pass (3700+ lines of Pascal) in about ten minutes.

Availability

The compiling system will be made available to any non-profit organisation, for the cost of the distribution, from:

Computer Centre
Massey University
Palmerston North
New Zealand.
Although no support for the system can be provided by the Computer Centre, rough implementation notes and advice are available from the author:

N. S. James
Computing Centre
University of Otago
P.O. Box 56
Dunedin
New Zealand.

10 January 1980

IBM 370 (Stony Brook)

From the release note accompanying Release 3.0:

"....... Release 3.0 of the Stony Brook Pascal/370 compiler completes the implementation of Pascal files (for the production version), as well as correcting a few errors reported in Release 2. All further maintenance will be relative to Release 3.0, so it should be installed immediately. If you have presently a Release 2 or Release 1 distribution tape, please return it to:

Ms. Patricia Merson
Department of Computer Science
SUNY at Stony Brook
Stony Brook, New York 11794

"....... Fairly detailed internal documentation for Pass 2 and Pass 3 of the Stony Brook compiler is now available on request from Ms. Merson. If you plan to perform any modifications of the compiler itself, you should obtain these documents. Pass 1 internal documentation has not yet been produced. ......."

(Machine-dependent details concerning internal versus external files follow.)
The basic material of the order consists of one copy each of the
Pascal/VS Language Reference Manual (SH20-6168) and the Pascal/VS
Programmer's Guide (SH20-6162). The machine-readable material con­sists
of source code, program load modules, and catalogued proce­dures. When ordering the basic material, specify one of the
following numbers

<table>
<thead>
<tr>
<th>Specify Number</th>
<th>Track Density</th>
<th>Description</th>
<th>Volume Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>9029</td>
<td>9/1600</td>
<td>Mag tape</td>
<td>None/DTR</td>
</tr>
<tr>
<td>9031</td>
<td>9/6250</td>
<td>Mag tape</td>
<td>None/DTR</td>
</tr>
</tbody>
</table>

Monthly charges for this licensed Installed User Program will not be
waived. The designated machine type is System/370.

Monthly charges shown above are provided for information and are
subject to change in accordance with the terms of the Agreement for
IBM Licensed Programs (Z120-2800).

DOCUMENTATION

The Pascal/VS documentation consists of:

<table>
<thead>
<tr>
<th>Document Name</th>
<th>Order Number</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pascal/VS Language Reference</td>
<td>SH20-6168</td>
<td>$16.50</td>
</tr>
<tr>
<td>Pascal/VS Programmer's Guide</td>
<td>SH20-6162</td>
<td>$12.50</td>
</tr>
<tr>
<td>Pascal/VS Reference Summary</td>
<td>GX20-2365</td>
<td>no charge</td>
</tr>
<tr>
<td>Pascal/VS Availability Notice</td>
<td>G320-6387</td>
<td>no charge</td>
</tr>
</tbody>
</table>

The Reference manual describes the Pascal/VS language. The Program­mer's Guide describes how to use the compiler in the OS/VSL, OS/VSp
and VM/CMS environments.

The documentation may be ordered through your local IBM branch
office.

MAINTENANCE

IBM will service this product through one central location known as
Central Service.

Central Service will be provided until otherwise notified. Users
will be given a minimum of six months notice prior to the discon­tin­uance of Central Service.

During the Central Service period, IBM, through the program
sponsor(s) will, without additional charge, respond to an error in
the current unaltered release of the compiler by issuing known error
 correction information to the customer reporting the problem and/or
issuing corrected code or notice of availability of corrected code.

However, IBM does not guarantee service results or represent or war­rant that all errors will be corrected.

Any on-site program service or assistance will be provided at a
charge.

Documentation concerning errors in the compiler may be submitted to:

IBM Corporation
555 Bailey Avenue
P.O. Box 50020
San Jose, California 95150
Attn: Larry B. Weber
Telephone: (408) 463-3159 or
Tieline: 8-543-3159

Marketing Sponsor

IBM Corporation
DPD, Western Region
3424 Wilshire Boulevard
Los Angeles, California 90010
Attn: Keith J. Warltier
Telephone: (213) 736-4645 or
Tieline: 8-285-4645

STANDARD

Pascal/VS supports the currently proposed International Standards
Organization (ISO) standard and includes many important extensions.
Among the extensions are:

- Entry and external procedures to provide separate compilation
- "Include" facility to provide a means for inserting source from
  a library into a program
- Varying length character strings, string concatenation, and
  string handling functions
- Static variables
- The "ASSERT" statement
- "LEAVE" and "CONTINUE" statements for more flexible loop control
- "OTHERWISE" clause on the CASE statement
- Subranges permitted as CASE statement "labels"
- Integer, real, and character constants may be expressed in
  hexadecimal
Various predeclared system-interface routines such as HALT, CLOCK, DATETIME, RETCODE, etc.

MEASUREMENTS

Under VM/CMS the compiler will compile a typical program of 1000 lines at the approximate rates shown below:

<table>
<thead>
<tr>
<th>Host System</th>
<th>Rate of compilation</th>
</tr>
</thead>
<tbody>
<tr>
<td>370/158</td>
<td>10,000 lines per minute</td>
</tr>
<tr>
<td>370/168</td>
<td>20,000</td>
</tr>
<tr>
<td>3033</td>
<td>40,000</td>
</tr>
</tbody>
</table>

If the compiler listing is suppressed, the performance improves by 20 to 25 per cent.

RELIABILITY

Prior to external release, the compiler was distributed to over 60 test sites within IBM. The first internal shipment of the compiler was in July of 1979. All errors reported prior to the release of the compiler have been corrected.

DEVELOPMENT METHOD

The compiler consists of two passes which run as two separate programs. The first pass is based on an extensively modified version of the Pascal P4 compiler (authored by Urs Ammann, Kesav Nori, and Christian Jacobi). The P4 compiler was re-targetted to produce U-code instead of P-code as an intermediate language. U-code is an enhanced version of P-code that was designed by Richard L. Sites and Daniel R. Perkins (Universal P-code Definition, U.C. San Diego, UCSD/CS-79/037, 1979). The compiler employs the error recovery algorithm described in A Concurrent Pascal Compiler for Minicomputers by Alfred C. Hartmann (Springer-Verlag, 1977).

The second pass of the compiler translates the U-code directly into an OS object deck. The translator performs local common subexpression elimination, local register optimization, dead store removal, removal of redundant checking code, removal of cascading jumps, and various peephole optimizations.

All but 5% of the execution library is written in Pascal/VS; the remainder is in assembler language. I/O and heap management is done by calls to Pascal procedures.

The compiler, written in Pascal/VS, is shipped with all run time checking enabled. The compiler eliminates unnecessary range checks by keeping track of the lower and upper bounds of expressions involving subrange variables. The checking code in the compiler costs only 3 to 10% in performance.

The development of Pascal/VS began in January, 1979. To bootstrap the compiler, an experimental Pascal compiler developed by Larry Weber was used; it was a one pass compiler written in PL/I (believe it or not!).

The first bootstrap was completed in June, 1979. Since that time, the compiler has been tested, enhanced, and modified to conform to the proposed ISO standard.

LIBRARY SUPPORT

Pascal/VS supports separate compilation of routines and uses standard OS linkage conventions. A Pascal/VS program may call routines written in FORTRAN, COBOL, and Assembler language.

DEBUGGER SUPPORT

Pascal/VS supports an interactive symbolic debugger which permits:

break points to be set
statement by statement walk-through of a procedure
variables to be displayed by name and in a form which correspond to their type (pointers, field qualifiers and subscripts are allowed).
Thank you for your inquiry about DYNASOFT PASCAL. I hope that this will answer most of your questions and help you decide if it will be a useful addition to your system.

DYNASOFT PASCAL was designed to make a practical subset of the PASCAL language available to the users of relatively small cassette-based 6800 and 6809 computers. Both versions occupy slightly less than 8K bytes and require at least 12K of continuous RAM beginning at $0020 to edit and compile programs of reasonable size. The compiler will compile itself in 32K, although the source code is not included in the package.

The 6800 version was designed for the SWTPC 6800 computer with the SWTBUG™ monitor, but it can be adapted to run with most other monitors with minor patching. The 6809 version is completely self-contained with its own embedded device drivers, and is independent of any particular monitor. Both versions include the compiler, p-code interpreter, and a line oriented text editor, and are priced at $35.00. They are supplied on a Kansas City Standard cassette in Motorola "81" format at 300 baud, and come with a 32 page user's manual.

The 6800 version is also available in ROM, intended for use with the SWTBUG™ monitor on the SWTPC HP-A2 processor board. It occupies the 8K block at $C000 and is supplied in four TMS2516 EPROM's. The price is $300.00. We do not keep a stock of blank ROM's, so please allow 8 weeks for processing.

All orders should include $3.00 for postage and handling. Payment can be made by postal money order, check, or VISA account in either Canadian or U.S. funds.

Thank you again for your interest.

Allan G. Jost, Ph.D.
DATA TYPES:

- INTEGER (16 bit)
- CHAR (8 bit)
- BOOLEAN
- ARRAY (one dimensional)
- scalar (user defined)
- subrange
- pointer

ARITHMETIC AND LOGICAL OPERATORS:

+ - * DIV MOD AND OR NOT

RELATIONAL OPERATORS:

= <> < > <= >=

LANGUAGE FEATURES:

- CONST
- TYPE
- VAR
- PROCEDURE
- FUNCTION
- IF-THEN-ELSE
- BEGIN-END
- WRITE
- READ
- Machine-language subroutines with parameters
- 80 character identifiers (first 4 unique)
- Absolute memory addressing using pointers
- LINK to other programs
- Full recursion

PREDEFINED PROCEDURES AND FUNCTIONS:

- ODD
- SHR
- FIND
- HALT
- LINK
- MOVL
- MOVR
- SETP

SUPERVISOR COMMANDS:

- Load
- Save
- Edit
- Compile
- Go
- Move
- Quit

EDITOR COMMANDS:

- New
- Top
- Bottom
- Up
- Down
- Find
- Print
- Insert
- Kill
- Replace
- Quit

To Andy Mickel,
editor of Pascal News

Dear Andy,

Enclosed you find checklists of two Pascal implementations we made on Motorola microcomputers: an M6800 and an XC6800, which is the experimental version of the M6800.

The M6800 implementation has already been in operation for about 2 years now and during this period the system proved to be extremely reliable and stable. This system is internally known as the TOME system (Pascal on Motorola microcomputer equipment). The compiler generates a kind of Pcode which is quite different from the .codes of the portable Pascal compiler and the UCSD-code.

The compiler is not a Pascal derivative, but is written from scratch. The code generated by the compiler is interpreted without the interference of an optimizer, a linker or such. The interpreter is 3.5 Kbytes of machine code and the compiler 17.6 Kbytes of Pcode. Depending on the scattering of files on floppy disk the compilation speed is between 400 and 600 lines per minute.

The language implemented contains the proposed ISO Pascal standard as a subset. The only restriction is: files must be declared in the outerblock only (file parameters of procedures and functions are of course possible). The extensions include:

- a library facility (on source level)
- interfacing with assembly language routines
- absolute address specification of variables (to allow memory-mapped I/O without the need of assembly code)
- subroutines and OTHERWISE as labels in a case-statement, subroutines also in the variant-part of records
- if the program contains a record-type definition like
  complex = RECORD re, im: real END
  then the construct complex(x,y) is an expression of type complex provided x and y are of type real.
- the so-called "bounless" array parameters.
- in addition to AND and OR the short-circuited NAND and NOR.
- random-access files.
- interactive I/O via files input and output

The compiler will always select the most compact representation of sets (up to 16 bytes). Hence sets of characters are possible. Furthermore a
SET OF 0..? requires only one byte and can beautifully be used to communicate with peripherals, due to the memory-mapped I/O.

If programs are run with runtime checks included then the detection of an
error will result in a symbolic dump of the program's stack, including identifiers
of variables and procedures, and line numbers of the error and "current"
procedure calls. Various errors not normally checked for will be detected in case
the runtime checks are turned on, e.g. a student-proof method to check changes
of a controlled variable in a for-statement.

In order to speed up some of the clerical tasks of the interpreter, some
IC's were added to the processor. The processor board, however, is still
compatible with the original Motorola 68000 bus. The additions allow for
a continuous check on stack overflow, a check which, when done in software,
is time-consuming and/or difficult (the P60 and UCSD strategy are unscalable).

The POMME system normally operates in a single-user environment with an
EXR/ceiver or EXClterm and a dual floppy disk drive. It is, however, possible to
interconnect up to 6 of these systems to form a multi-user system, sharing
the disk space. The POMME system will then guarantee mutual exclusion on R/W
access, on the basis of individual sectors.

One of the programs available on the POMME system is a cross-
compiler for the XC 68000. This compiler (reals and files are not yet
implemented) generates relocatable machine code which does not require an
interpreter, runtime package or operating system to execute. The code is
close to optimal and to achieve this the compiler does not consist of a
single pass but is a 3-pass compiler. This process necessarily slows
down compilation, mainly because all intermediate code is kept on a
floppy disk. The output of the compiler need not be input to an assembler
but is executable, position independent code.

Although I have written all software of the POMME system it is now
maintained and distributed by

EPOS (Efficient Pascal Oriented Systems)
Generaal de Carivalaan 60
5623 GL Eindhoven
The Netherlands
tel. 040 - 445552

Some sample programs were run for speed comparisons. Roughly speaking, the
H6000 system compiler at about 4 times and executes at about twice the
speed of UCSD implementations on LSI-11 and Z-80. We feel this pretty
impressive for a 1 MHz 8 bit processor. The cross compiler for the XC 68000
is much slower, it compiles at half the speed of LSI-11 and 2-3x UCSD.
Execution times, however, are about equal to DEC-10, half the speed of a
Burroughs B700 and a quarter of the speed of CDC Cyber 175. Notice
that the XC 68000 is only a prototype of the M68000 running at half
the projected speed.

Finally it should be noted that a compiler for the H6800 along the lines
of the XC 68000 implementation will be available soon.

Yours sincerely,

JL van de Snepvogel
Eindhoven University of Technology
Dept. of Mathematics
March 19, 1980
Checklist Motorola M6800 (POMME system)

date 1980, March 19

maintainer/distributor EPOS
Generaal de Carislaan 60
5623 GL Eindhoven (The Netherlands)

maintenance fully maintained

standard contains standard Pascal as a subset

measurements roughly twice the speed of the UCSD-implementations on LSI-11 and Z-80; compilation times even four times.

reliability 2 years in operation, very stable and reliable

library support source libraries in Pascal
linkage to assembly language routines

machine Motorola M6800

Zilog Z-80 (MetaTech)

(See the checklist in issue #17 under Intel 88000/8885 (MetaTech))

Zilog Z-80 (Digital Marketing)

This compiler runs under CP/M and is a Pascal-P descendant. The price is $350.

Digital Marketing
2670 Cherry Lane
Walnut Creek, CA 994596
People's Software at nonprofit Computer Information Exchange is selling a tiny Pascal compiler for $15.

Written in Basic, People's Pascal I runs on any 16K TRS-80 Level II system. Compilers let computerists write fast, efficient machine code while working with a higher-level language. Pascal is the structured language everyone is talking about—and studying in college.

The People's Pascal I program development system comes on a tape with 14 programs, and 18 11x17" pages of documentation. Programs include editor/compiler, interpreter, translator, run-time system, and two demonstration programs.

People's Pascal I compiler produces P codes, which the translator converts to Z-80 code, the TRS-80 native language. The user is given the option of optimizing for either speed or memory efficiency. Programs written via People's Pascal I run three times faster than those in Level II Basic—graphics is eight times faster.

To produce object programs, the computerist must use the People's Pascal I programs, plus Tandy T-Bug. Use of Tandy editor/ assembler is optional.

The People's Pascal I program development system, with editor/compiler and interpreter written in Basic, and its multiple parts, is not the ultimate in speed and simplicity of use.

People's Pascal II, at $23, is easier to use and faster operating. It is all one machine-language program. Programs written in Pascal II do not execute quite as fast as those in Pascal I because the system does not produce Z-80 object programs of the user's source program.

Both Pascal I and II compile user programs into P-codes. Both systems work in an interpretive mode, interpreting P-codes into Z-80 codes.
IMPLEMENTATION NOTES ONE PURPOSE COUPON

0. DATE

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

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

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

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

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

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

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

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

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

10. LIBRARY SUPPORT (* Any other support for compiler in the form of linkages to other languages, source libraries, etc.*)
NOTE: Pascal News publishes all the checklists it gets. Implementors should send us their checklists for their products so the thousands of committed Pascalers can judge them for their merit. Otherwise we must rely on rumors.

Please feel free to use additional sheets of paper.