PASCAL USER'S GROUP

USER'S GROUP

PASCAL NEWSLETTER

NUMBER 8

COMMUNICATIONS ABOUT THE PROGRAMMING LANGUAGE PASCAL BY PASCALERS

MAY, 1977

TABLE OF CONTENTS

RENEW!!

PASCAL USER'S GROUP POLICIES

Purposes - are to promote the use of the programming language Pascal as well as the ideas behind Pascal. Pascal is a practical, general purpose language with a small and systematic structure being used for:

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

Membership - is open to anyone: particularly the Pascal user, teacher, maintainer, implementor, distributor, or just plain fan. Institutional memberships, especially libraries, are encouraged. Membership is per academic year endi June 30. Anyone joining for a particular year will receive all 4 quarterly issues of *Pascal Newsletter* for that year. (In other words, back issues ar sent automatically.) First time members receive a receipt for membership; renewers do not to save PUG postage.

Cost of membership per academic year is $4 and may be sent to:
Pascal User's Group/ %Andy Mickel/University Computer Center/227 Exp Engr/ University of Minnesota/Minneapolis, MN 55455 USA/ phone: (612) 376-7290

In the United Kingdom, send £2.50 to:
Pascal Users' Group/ %Judy Mullins/Mathematics Department/The University/ SOUTHAMPTON/SO9 5NH/United Kingdom/ (telephone 0703-559122 x2387)

PASCAL NEWSLETTER POLICIES

The *Pascal Newsletter* is the official but informal publication of the User's Group. It is produced quarterly (usually September, November, February, and May). A complete membership list is printed in the November issue. Single back issues are available for $1 each. Out of print: #s 1,2,3,4 *

POLICY

The contribution by PUG members of ideas, queries, articles, letters, and opinions for the *Newsletter* is important. Articles and notices concern: Pascal philosophy, the use of Pascal as a teaching tool, uses of Pascal at differe computer installations, portable (applications) program exchange, how to promote Pascal usage, and important events (meetings, publications, etc.).

Implementation information for the programming language Pascal on different computer systems is provided in the *Newsletter* out of the necessity to spread the us of Pascal. This includes contacts for maintainers, documentors, and distributors of a given implementation as well as where to send bug reports Both qualitative and quantitative descriptions for a given implementation a publicized. Proposed extensions to Standard Pascal for users of a given implementation are aired. Announcements are made of the availability of ne software writing tools for a Pascal environment.

Miscellaneous features include bibliographies, questionaires, and membership lists. Editor's notes are in Pascal style comments (**).

ALL THE NEWS THAT FITS, WE PRINT. PLEASE SEND WRITTEN MATERIAL TO THE NEWSLETTER SINGLE SPACED AND IN CAMERA-READY FORM. USE NARROW MARGINS; (LINE WIDTH 18.5 CENTIMETERS). REMEMBER, ALL LETTERS TO US WILL BE PRINTED UNLESS THEY CONTAIN A REQUEST TO THE CONTRARY.

- Andy Mickel, editor, John P. Strait, associate editor, April 26, 1977.

UNIVERSITY OF MINNESOTA
TWIN CITIES

University Computer Center
227 Experimental Engineering Building
Minneapolis, Minnesota 55455

(612) 376-7290

his issue with so many important topics is late. I think that George Richmond deserves nother round of thanks for the early work he did on Pascal Newsletter. With this, the ourth issue I've done, I have to say that it is a lot of work. Without Sara Graffunder nd Jim Miner, who edited the Here and There and Implementation Notes sections respectively, his issue would not have appeared.

RENEW

e've lowered the cost of PUG membership by keeping the price the same ($4_{1977} < 4_{1976})!

his is the last (and first) renewal notice you'll get. Please renew, especially if you hink we are doing some good in the world. If you are not reading your own copy of the ewsletter, why not help us out: join for yourself (we need more members to keep the price he same). Just think of it as giving up eating out one night in the next year. And we on't refuse additional (no strings attached) contributions!

TANDARDS

ee the Open Forum section for a series of letters.

ICROPROCESSOR Pascal

ee the Here and There News section under Charles Bacon, P.M. Lashley, Steve Legenhausen, ndy Mickel, David A. Mundie, and see Implementation Notes under both "Comment: Microproc- ssors" and under individual specific manufacturers's names. And Ken Bowles's article.

ascal Newsletter #9.

eadline for written contributions is July 15. Changes in POLICY: #4 is now out of print. ll written material must now be single spaced and typed with narrow margins. We are unning out of room!

HIS ISSUE (#8)

nfortunately we have had to cut material from this issue ("all the news that fits..."). eorge Richmond sent a 5 page bibliography which we couldn't find room for. It had only 5 new entries over his last one in #4, and is incomplete these days if you keep up with ascal Newsletter. We were also unable to print a Roster increment as we did in #7. regret this because it is the roster which enables Pascalers to get together especially f they are in the same area. This time the number of new members totals 345! It would ave taken 6 full pages to print in a new compressed format! We just couldn't afford it. le also had to reformat every contribution to save space, and omit extraneous material.

ut, we have no shortage of material (unlike the disease which afflicted the FORTRAN ulletin, the LISP Bulletin, the SNOBOL Bulletin, etc.).

le have had many suggestions regarding the newsletter. We want to keep it informal and interesting and prevent its degeneration into a slick, useless, "professional" journal.

UG and Pascal Newsletter Mechanics

UG now has 943 members in 29 countries and 47 states! We need more members to stay inancially solvent (we are currently in the black, barely) and we need them as well as enewals early in the academic year (preferably before August 15). I now strongly disagree ith my earlier idea (and Mike Hagerty's letter in this issue) of becoming affiliated with CM (like STAPL under SIGPLAN). Did you know that according to Garth Foster (January 6, 977) STAPL (SIGPLAN Technical committee on APL) only had 973 members after more than 5 ears in existence? If we affiliated with ACM, the price would probably double, but we'd e compensated with fancy letterhead on the stationary.

EDITOR'S CONTRIBUTION

PUG has a broad base with many non-academic members. We have kept the price low, publicized PUG in unconventional ways (unlike ACM) and in the process have become known in industry where the real changes can be made. We just completed our fourth mass mailing (350) on March 28 to the holdouts from George Richmond's old mailing list from newsletters 1-4.

I would like to encourage all PUG members to use their imaginations in making Pascal and PUG more visible. Write letters to the editor of popular trade journals such as COMPUTERWORLD, DATAMATION, COMPUTING EUROPE, etc. Distributors of compilers should send an All Purpose Coupon to each recipient of their implementations. Write to SIGCSE (Pascal's strong point is Computer Science Education). I can't do all of these things.

I've noticed some big discrepencies in PUG membership at several universities which have a fair amount of Pascal users. It seems that some local people have not done all they can to tell their users about PUG. Why is it for example that at the University of Minnesota there are 48 PUG members, at Lehigh University 13, at Indiana University, the University of Texas, and the Technical University of Berlin 7, and at the University of Illinois, Georgia Tech, the University of Southwestern Louisiana, Cornell, and the Imperial College London, etc. there are 6 PUG members while at the University of Colorado there is only 3, the University of Washington only 2, and at the University of Manitoba, SUNY Buffalo, and the University of Massachusetts only 1?

BACK ISSUES

I'm sorry that we are slow, but we are not in the publishing business. As I stated in #7 we have had terrific growing pains resulting from not realizing back in September how popular PUG was going to be. We are temporarily out of print with #5 and this holds up mailing 5,6,and 7 to new members as we cannot afford postage for separate mailings. As it is, it is very expensive to mail back issues. At PUG "central" here in Minnesota, we have no secretaries. John and I (with help from people like Sara and Jim) have opened all our own mail, answered with personal notes all inquiries, handwritten most addresses on envelopes, handled all the typing, mailing of back issues, filing, accounting, the mailing label data base; and sent invoices and bills to persons who haven't paid. That's right, we never planned on some people not paying. Those who still owe PUG money are: Bengt Norstrom, Lars Magnusson, Bernhard Nebel, Roland F. Blommer, Stanley B. Higgins, Karl J. Astrom, Wayne Fung, John S. Sobolewski, T. Hardy, Ada Szer, and John Nolan. This is as of today, and I wouldn't be surprised to see their money soon, and I don't in any way want to imply that each does not eventually intend to pay!

SUMMARY

I want to thank all of those who have helped this year, especially Judy Mullins, David Barron, Carroll Morgan and Tony Gerber (who have enabled Australasian re-mailing with zero compensation) and Teruo Hikita for remailing #7 to Japanese members. Finally many thanks are due to the University Computer Center here at the University of Minnesota, particularly Peter Patton, our director, and Lawrence Liddiard our associate director for systems for enabling PUG to thrive.

Andy

- April 26, 1977

HERE AND THERE WITH PASCAL

NEWS (ALPHABETICAL BY LAST NAME)

Charles Bacon, 10717 Burbank Dr., Potomac, MD 20854 (PUG member): "I am interested in a Pascal running on a RSX-11M system as well as on the KI-10....also on any 8080 system." (* 1/10/77 *)

Mark Becker, 300 Collingwood Ave., Fairfield, CT 06432 (PUG member): "I'd like to locate a version of PASCAL for the PDP 11 that does not use or require the Floating Point Processor." (* 1/31/77 *)

(* From the newsletter of the University Computer Center at the University of Southern California, 1020 W. Jefferson Blvd., Los Angeles, CA 90007: UCC has added several JCL procedures (for its IBM 370 system) so that users can invoke the University of Manitoba version of Pascal. The procedures perform one-step monitor; compile; compile, load and go; compile, linkedit; compile, linkedit, and go; load and go; linkedit and go; and compile and punch an object deck. 1/1/77 *)

Gary Boos, 517 N. 7th St., Bismarck, ND 58501 (PUG member): "I am interested in knowing about chess programs written in Pascal."

Kevin W. Carlson, 1531 Simpson St. Madison, WI 53713 (PUG member): (* Wants to know if there is a group of Pascal users in or near Madison. 2/9/77 *)

C. R. Corner, 514 S. 9th St., Moorhead, MN 56560 (PUG member): "I'm trying to implement Pascal on the PDP-8 and on the PDP-11. Any suggestions?" (* 3/1/77 *)

Frederick C. Cowan, The Aerospace Corporation, Mail station A2-2043, P. O. Box 92957, Los Angeles, CA 90009 (PUG member): "I am interested in the mods to make [release 2 of PASCAL 6000-3.4] run on the 7600 under Scope 2.2." (* 3/18/77 *)

Mattia Hmeljak, Ist. di Elettrotecnicà ed Eletronicà, Università di Trieste, Trieste, Italy (PUG member): "In Trieste University a CDC computer exists and a Pascal compiler is implemented there.

We have also an HP-2100 mini-computer and we would like to run some programs there for teaching and for research. For these reasons we intend to implement the Pascal compiler on this machine.

As a first step . . . we intend to write a P-code interpreter using the Pascal-written interpreter and translating it into H-P Algol. Therefore we would be glad to know if someone else is working to implement Pascal on the same mini-computer. . . . We thank you also for any information you will consider useful to give us for our work." (* 2/5/77 *)

Stanley B. Higgins, Dept. of Medicine, Vanderbilt University, Nashville, TN 37232 (PUG member): ". . . our group operates a PDP-11/40, PDP-11/34 and a PDP-11/55 . . . software . . . by DEC . . . RT-11 and RSX-11M operating systems. . . . We would be most interested in knowing of [Pascal compilers]." (* 2/23/77 *)

Robert L. King, 1452 Sandra Dr., Endicott, NY 13760 (PUG member): "If possible, please forward information on free or very inexpensive Pascal compilers for an IBM 370/178 under VSI with 3330's and 9-track tapes." (* 2/1/77 *)

Joseph Lachman, Computer Center, University of Illinois at Chicago Circle, Box 4348, Chicago, IL 60680 (PUG member): " . . . At present the UICC computer center has no Pascal compiler. Any advice you could offer us relative to the availability, quality, and costs of PASCAL compilers that will run on IBM/370 or DEC PDP-11 computers would be greatly appreciated." (* 4/5/77 *)

J. Larmouth, Director, Computing Laboratory, University of Salford, Salford M5 4WT, England (PUG member): " Having moved to Salford from Cambridge, I have ceased work on Pascal. Unfortunately there was nobody available at Cambridge to continue the work, so that our efforts towards a 370 implementation should be considered abandoned.

"We did produce and distribute an interpreter system but Cambridge . . . does not have the man-power to continue even this service.

"Sorry this is all so negative. My interest in Pascal remains, although you might be interested to know that I am perhaps more interested in EUCLID, as would, I think, be most members of PUG if they knew more about it." (* 1/5/77. For information about Euclid, consult B. W. Lampson, et. al., "Report on the Programming Language Euclid," SIGPLAN Notices , 12:2 (February 1977); and G. J. Popek, et. al., "Notes on the Design of EUCLID," SIGPLAN Notices, 12:3 (March 1977), 11-19. *)

P. M. Lashley, Director of Computing CSCS, POB 764, 114 S. Bullard St., Silver City, NM 88061: (* From a letter to the editor of Byte, 2:2 (February 1977), 77-78 *) "I write primarily in response to Mr. Skye's letter in your August issue. I can only conclude that he had been with IBM too long, otherwise he would not attempt to debase the 8080 with FORTRAN or PL/1. FORTRAN is a virtual pterodactyl, flying solely by inertia, whereas PL/1 is much better, but too rambling in construction. If he indeed takes up the admirable task of writing a high level compiler for the 8080, he would be better advised to base his compiler on a fully structured language such as PASCAL." (* The letter goes on for several paragraphs. *)

Steve Legenhausen, 12 Barnard Street, Highland Park, NJ 08904 (PUG member): "I think it is absolutely important that persons promoting Pascal realize the danger of BASIC's becoming the permanent and only language on microprocessors. One only has to pick up any issue of the computer hobbyist magazines such as Dr. Dobbs Journal, Byte, Kilobaud, Creative Computing, etc., to find that each is filled with BASIC. Some effort should be put forth to promote Pascal in this medium." (* 12/31/76 *)

Chris P. Lindsey, Computing Coordinator, Harvey Mudd College, Claremont, CA 91711 (PUG member): "Do you know of a well-documented, error free version of PASCAL which runs on a DECsystem-10 with a KA processor?" (* 1/77 *)

R. A. Lovestedt, 20427 St 192, Renton, WA 98055 (PUG member): "Any PASCAL work on an HP3000?" (* 2/10/77 *)

William Lyczko, Software Development, NCR Corporation/Terminal Systems, 950 Danby Road, Ithaca, NY 14850 (PUG member): "I am interested in any information you may have on implementation of PASCAL for microprocessors." (* 1/7/77 *)

Philip J. Malcolm, former address Zeus-Hermes Consultants Ltd., Shropshire House, 2-10 Capper Street, London; new address c/o Bank of Adelaide, 11 Leadenhall St., London EC3V 1LP, England (PUG member): "Zeus-Hermes is . . . investigating the possibility of adopting a Pascal -- or Modula -- type language for in-house development of mini- and micro-computer software across a broad range of target machines.

"Ideal would be a compiler:
written in its own source language; and
executable on a microcomputer (with say 32-64K bytes of RAM, diskettes); and
easily transportable to different target machines; and
relying on a very small run-time monitor/support package.

"We would be delighted to hear from those possessing or working towards such a system." (* 1/3/77 *)

Andy Mickel, Univ. Computer Center, 227 Exp. Engr., U. of Minnesota, Minneapolis, MN 55455 (PUG member) reports receiving an educational questionnaire from Intel about computer courses and micro-processors. The question, "What programming languages are used?" contained the check-off answers Fortran, Algol, PL/1, PL/M, Basic, and Pascal. Not included were Cobol, Lisp, Snobol, etc. (* Andy's response to the catch-all question at the end was, "When are you going to support Pascal or a Pascal-subset and give up on Basic?" *)

David A. Mundie, French Department, 302 Cabell Hall, University of Virginia, Charlottesville, VA 22903 (PUG member): "Is Zilog really making a microprocessor that executes PASCAL constructs as its machine-level language (Byte, v.2, no. 4, April 1977, p. 140)?" (* 4/3/77 Will a PUG member please write Zilog to ask, then send the answer to the newsletter? *)

Mark O'Bryan, Computer Center, Western Michigan University, Kalamazoo, Michigan 49001: ". . . I'm in charge of PASCAL implementation and maintenance at WMU. We have an old version of NAGEL's compiler for the PDP-10 and will be releasing it for use here in early March . I"ll keep you informed on user reaction when it happens.

Gene H. Olson, 421 County Road 3, Apt. 512, Hopkins, MN 55343 (PUG member): "The best argument against formatted reads has yet to appear in the PUG newsletter. In processing large amounts of formatted data (the supposed rationale of formatted reads) keypunch or similar errors cause both formatting and content errors which render formatted reads useless. In other words, in a production environment, the program must check data character-for-character as it is coming in." (* 2/25/77 *)

Jerry L. Ray, 21320 Oldgate Rd., Elkhorn, NE 68022 (PUG member): "I am attempting to sell the idea of using PASCAL instead of FORTRAN as a first language in a Computers & Business course. Any information to support my argument (institutions using PASCAL, etc., as well as the structure aspects) would be greatly appreciated." (* 4/8/77 *)

R. Waldo Roth, Computer Science Dept., Taylor University, Upland, IN 46989 (PUG member): ". . . I would also like to know about the availability of a PASCAL package to run on DEC 11 systems under RSTS or RT-11." (* 2-24-77 *)

Carl W. Schwarcz, Digital Equipment Corp., MR 1-2/E27, 200 Forest Street, Marlboro, MA 01752 (PUG member): ". . . While employed by Control Data I was responsible for the design and implementation of two compilers for a Pascal-based programming language ('the Software Writers' Language') for the Cyber 170 and Cyber 270." (* 1/25/77 *)

Arthur I. Schwarz, Hughes Aircraft Co., Bldg. 150/MS A222, Culver City, CA 90230 (PUG member): "Our installation is currently interested in gaining some expertise in using PASCAL. We would like to obtain a compiler for use on our Sigma 9 computer, or lacking this, a compiler with accessible code generators for either the CDC or IBM computer lines." (* 2/8/77 *)

Wayne Seipel, Box 8259 U.T. Station, Austin, TX 78712 (PUG member): "The University of Texas Computer Science department needs a PASCAL compiler for a [Data General] Nova 3D. The department has just purchased 2 processors, each with 32K words of memory and a 10Mega-byte disk. These will be used by both graduate and undergraduate students in a hands-on environment. Current plans call for the development of an operating system, and a PASCAL compiler would make life orders of magnitude easier. Any information on a compiler (completed, standard, PASCAL1, or PASCAL2) will be greatly appreciated.

Contact either James Peterson, Computer Science Dept., University of Texas, Austin, TX 78712, or myself." (* 3/14/77 *)

Kevin Weiler, 147 Cornell Qrtrs., Ithaca, NY 14850 (PUG member): "Has anyone implemented PASCAL/J on a PDP 11/45? (Is a Janus interpreter available?)" (* 1/21/77 *)

Nicholas Wybolt, 576 Leo Street, Hillside, NJ 07205 (PUG member): "Here at NJIT, Pascal is beginning to be used in a junior-level course in algorithms and data-structures; there is also individual interest in Pascal among faculty members and the student body.

"The student branch of the ACM is attempting to act as a medium of information in this matter. We are interested in your group and any related publications and activities. . . ." (* 2/4/77 *)

(* From a press release by the U. S. Department of Defense distributed by the British Computer Society, March 22, 1977, on "The U. S. Department of Defense High Order Language Effort," to reach a consensus on a common high order language for embedded systems, p. 8 *):

"Without exception, the following languages were found by the evaluators to be inappropriate to serve as base languages for a development of the common language: FORTRAN, COBOL, TACPOL, CMS-2, JOVIAL J73, JOVIAL J3B, SIMULA 67, ALGOL 60, and CORAL.

"Proposals should be solicited from appropriate language designers for modification efforts using any of the languages, PASCAL, PL/I, or ALGOL 68 as base languages from which to start. These efforts should be directed toward the production of a language that satisfies the DoD set of language requirements for embedded computer applications."

HERE AND THERE WITH PASCAL

CONFERENCES

International Federation of Information Processing Societies (IFIP), August 8-12, 1977 in Toronto. (* Would a PUG member who is there organize and publicize a Pascal User's Group gathering. We would, but we won't be there. Also, send in a resume of the meeting for Newsletter No. 9. Thanks. *)

ACM '77, Seattle, Washington, October 17-19, 1977. (* The same here for Newsletter No. 10.*)

REPORT on the Third Annual Computer Studies Symposium at Southampton (March 24-25)

"PASCAL - THE LANGUAGE AND ITS IMPLEMENTATION"

A little over halfway in this whirlwind, 48 hour happening, the medieval banquet began. David Barron (the baron) and Judy Mullins (the baroness) enjoyed the honor of reigning over and hosting the attendees; it was a delightful time indeed.

And so was the whole symposium! I must commend Judy for organizing the symposium down to the last detail and thank David for making it a reality. It was a success by several different measures. Around 134 persons attended. The proceedings officially listed (including speakers and last minute replacements): Austria 3; Belgium 4; Canada 1; Denmark 7; France 4; Germany 16; Great Britain 72; Ireland 8; Netherlands 2; Sweden 9; Switzerland 5; and the USA 3; The proceedings contain the texts of all 11 presentations and will be published later this year (see Books section). All except Per Brinch Hansen's which will appear in an IEEE publication.

David Barron, U of Southampton, opened the symposium with a talk entitled "Perspectives on Pascal" which looked at the past, present and future and concluded with a call to join a "Society to Combat Well-meant Attempts to Change Pascal (SCWACP)."

Urs Ammann, ETH, Zurich, was introduced as the great-grandfather of all Pascal compiler writers and summarized his work over the last 6 years in "The Zurich Implementation."

Jim Welsh, Queen's U, Belfast, likewise introduced as the grandfather of Pascal compiler writers detailed development and performance of "Two ICL 1900 Pascal Compilers."

David Watt, U of Glasgow, presented an extensive description of "A Pascal Diagnostics System" for the ICL 1900 implementation.

Mike Rees, U of Southampton, presented a description of the Pascal compiler effort on the ICL 2970 underway for the past 9 months in "Pascal on an Advanced Architecture."

Judy Mullins, U of Southampton, did not dream up hypothetical architecture, but rather critically combined existing architectural features in designing "A Pascal Machine?"

The next day began with Per Brinch Hansen, U of S. California describing his "Experience With Modular Concurrent Programming" and his opinions of the future.

Pierre Desjardins, U of Montreal, substituted for Olivier Lecarme, U of Nice, and sketched an overview of "Pascal and Portability" issues.

Brian Wichmann,National Physical Laboratory, Middlesex, coalesced various aspects on "The Efficiency of Pascal" in comparison to other languages and in different environments.

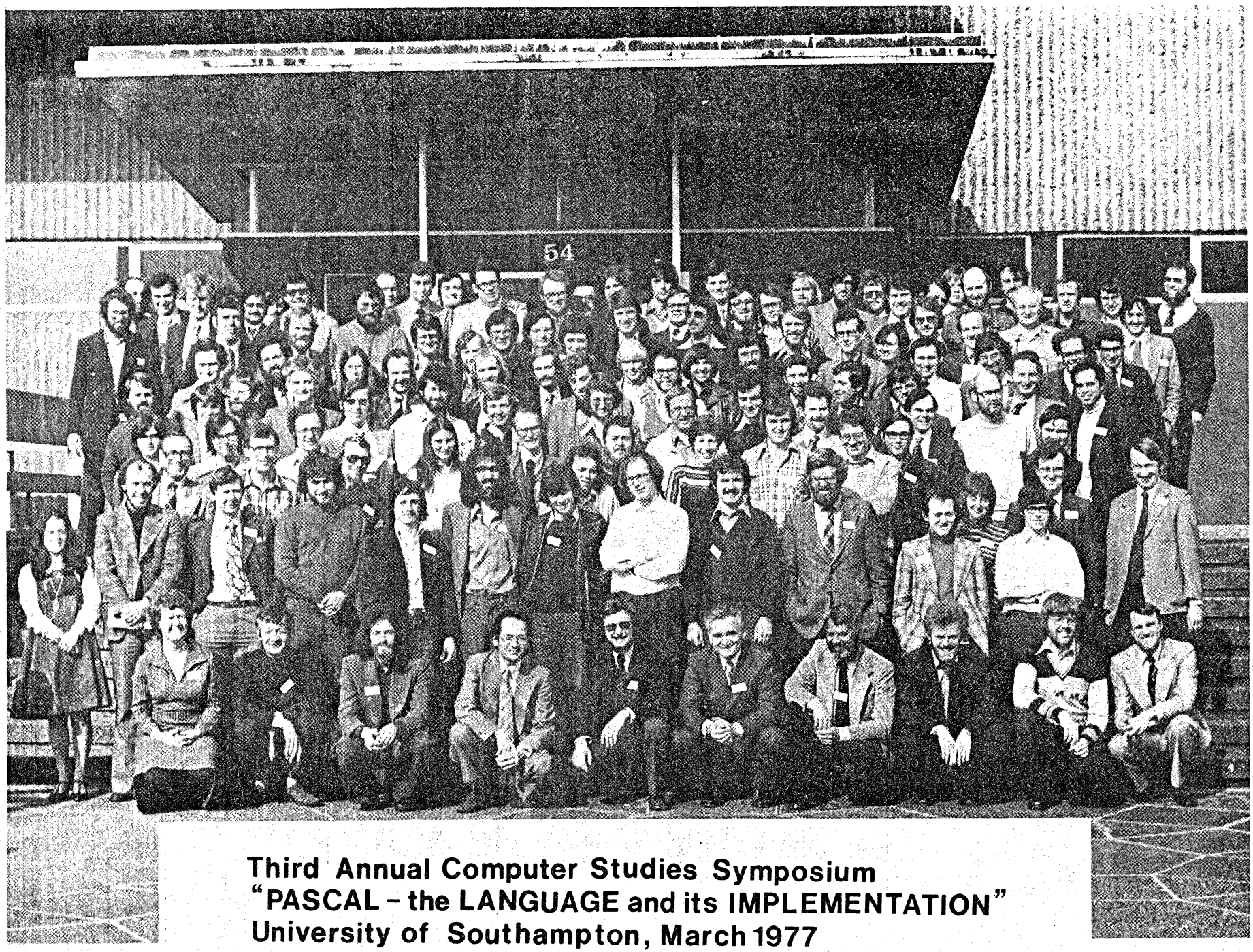
Graeme Webster, Teeside Polytechnic, advised others who introduce Pascal into the curriculum with a talk on "Pascal in Education."

There were two discussion sessions. Brian Wichmann led the first on "Pascal on Minis and Micros" and I introduced the second on "The Future of Pascal" concerning standards and extensions issues.

In between time, the opportunity to talk and argue with other Pascalers from so many places was a real treat for all, I'm sure. I managed to meet 48 people, and in the process confessed to Urs that it was hard to get used to intense, sudden exposure to so many cultural backgrounds.

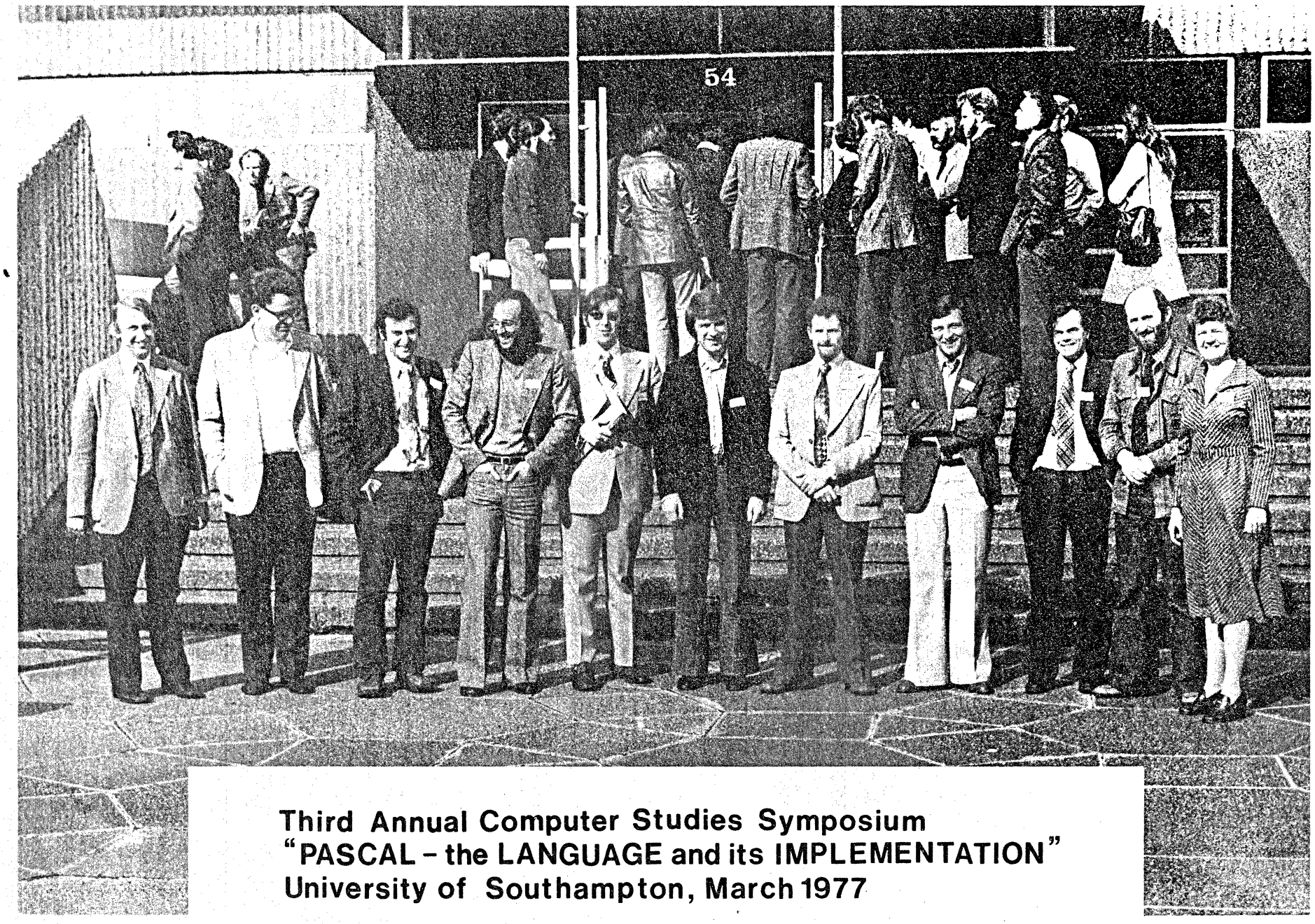
Perhaps the long range accomplishment of the symposium was to pass on a consensus to the rest of us in PUG regarding standards. See OPEN FORUM.

- Andy Mickel, April 17, 1977



Third Annual Computer Studies Symposium
"PASCAL - the LANGUAGE and its IMPLEMENTATION"
University of Southampton, March 1977

SYMPOSIUM ATTENDEES, (127 pictured here; not all names and faces known together!): A full



Third Annual Computer Studies Symposium
"PASCAL - the LANGUAGE and its IMPLEMENTATION"
University of Southampton, March 1977

SYMPOSIUM SPEAKERS, (pictured from left to right): David Barron, Per Brinch Hansen, Andy Mickel, Pierre Desjardins, Graeme Webster, David Watt, Mike Rees, Urs Ammann, Brian Wichmann, Jim Welsh, and Judy Mullins.

BOOKS AND ARTICLES

(* D. W. Barron, working with Rich Stevens, has offered to take over this section. What follows is a notice of the policy for the section, beginning with No. 9 *)

POLICY

In this section we shall try to keep PUG members up-to-date with the PASCAL literature under the general headings Languages, Textbooks, Implementation, Applications. At the least we shall give a brief citation of title, author and publisher. If possible we shall included a brief abstract and, if the importance warrants it, a critical review. In addition from time to time we shall give (hopefully) complete annotated bibliographies of selected areas: with the feedback from PUG members we should be able to build up a really comprehensive guide to the PASCAL literature.

Books and papers in the established journals are fairly easy to keep track of, but internal reports present much more difficulty. If you (or your institution) produce a report that you are willing to circulate, please send me a copy of the title page, or better still a copy of the report. The address is:

David Barron
Pascal User's Group (U.K.)
Department of Mathematics
The University
SOUTHAMPTON, SO9 5NH
U.K.

or W. Richard Stevens
Kitt Peak National Observatory
P. O. Box 26732
Tuscon, AZ 85726
U.S.A.

As with the rest of PUG, the success of this enterprise will rest largely on the enthusiasm and help of the membership.

10 February 1977 David Barron

(* The policy begins with the next issue. What follows is our new information about books and articles, and a review. *)

BOOKS

A Concurrent Pascal Compiler for Minicomputers, by Al Hartman, to be published by Springer-Verlag as Volume 50 in their Lecture Notes in Computer Science. Probably available by the end of April 1977. (* Al writes that the book will be of especial interest to ". . . any of your membership using the Concurrent Pascal or Sequential Pascal compilers developed at Caltech for the PDP-11/45 minicomputer." *)

Introduction to Computer Science, by Ken Bowles (U. of California, San Diego), to be published by Springer-Verlag in October 1977. (* The book is computer graphics oriented and uses Pascal as the teaching vehicle. Note the change of title from our citation in No. 5. *)

Introduction to Programming and Problem Solving with PASCAL, by G.M. Schneider, D. Perlman, and S. Weingart, to be published in hardback by Wiley and Sons in January 1978. A camera-ready manuscript of the book can be obtained by writing

Gene Davenport, Editor
John Wiley and Sons Publishers
605 Third Avenue
New York, NY 10016

The manuscript may, with written permission, be duplicated for class use until the publication of the book.

Pascal--the Language and its Implementation, proceedings of the Symposium in Southampton, March 24-25. (* At press-time, there is as yet no definite publisher and publication date. Perhaps details will be settled in time for publication in No. 9 *)

Structured Programming and Problem Solving with PASCAL, by Richard Kieburtz, Department of Computer Science, SUNY at Stony Brook, Stony Brook, NY 11794, to be published by Prentice-Hall sometime in 1977. (* This is rumored; we aren't sure of the title, etc. We hope we'll have the facts in time for No. 9. *)

ARTICLES

"Efficient Implementation and Optimization of Run-time Checking in Pascal," by Charles N. Fischer and Richard J. LeBlanc, SIGPLAN Notices, 12:3 (March 1977); 19-24.
(* From the abstract *): "Complete run-time checking of programs is an essential tool for the development of reliable software. A number of features of the programming language PASCAL (arrays, subranges, pointers, record variants (discriminated type unions), formal procedures, etc.) can require some checking at run-time as well as during compilation. The problem of efficiently implementing such checking is considered. Language modifications to simplify such checking are suggested. The possibility of optimizing such checking is discussed."

"Proceedings of the All-Union Symposium on Implementation Techniques for New Programming Languages, Novosibirsk 1975."
(* This publication came to us from David Barron, who received it from PUG member S. Pokrovsky, Computing Centre, USSR Academy of Sciences, Novosibirsk 630090, USSR. Most of the articles are in Russian, but the number of bibliographical references to publications about Pascal lead us to believe that the articles might be of interest to PUG members. Would someone who reads Russian (easily) volunteer to read and abstract the relevant articles for No. 9? We'll send a copy of the journal to you if you write to us in Minneapolis. The abstracts could go to David Barron for the next newsletter. *)

"Programming Languages: What to Demand and How to Assess Them," by Niklaus Wirth, Berichte des Instituts fur Informatik, E. T. H. Zurich, No. 17 (March 1976), 1-24.
(* From the abstract *): "The software inflation has led to a software crisis which has stimulated a search for better methods and tools. This includes the design of adequate system development languages.
This paper contains some hints on how such languages should be designed and proposes some criteria for judging them. It also contains suggestions for evaluating their implementations, and emphasizes that a clear distinction must be made between a language and its implementation. The paper ends with concrete figures about a Pascal implementation that may be used as yardstick for objective evaluations."

An Extract from "Professor Cleverbyte's Visit to Heaven," by Niklaus Wirth, Berichte des Instituts fur Informatik, E. T. H. Zurich, 17 (March 1976), 25-31. (* To appear in Software Practice and Experience *)
(* From the abstract *): "The following fable is a grotesque extrapolation of past and current trends in the design of computer hardware and software. It is intended to raise the uncomfortable question whether these trends signify real progress or not and suggests that there may exist sensible limits of Growth for software too."

"The Software Development System," by C. G. Davis and C. R. Vick, IEEE Transactions on Software Engineering, 3:1 (January, 1977), 69-84. (* A summary by PUG member Nick Solntseff, who sent in the citation *): Implementation of PDL-2, an extension of Pascal, to, among other things, include concurrent processing. Are also writing an OS in PDL-2.

"Some High-Level Language Constructs for Data of Type Relation: An Investigation based on Extensions to Pascal," by Joachim W. Schmidt, Bericht Nr. 31, Institut fur Informatik, Hamburg, January 1977.
(* From the abstract *): "For the extension of high-level languages by data types of mode relation, three language constructs are proposed and discussed:
- a repetition statement controlled by relations
- predicates as a generalisation of boolean expressions
- a constructor for relations using predicates.
The language constructs are developed step by step starting with a set of elementary operations on relations. They are designed to fit into PASCAL without introducing too many additional concepts." (* These extensions, which process relational data bases, are being experimentally implemented in Nagel's DEC-10 compiler at Hamburg *)

BOOK REVIEW

INTRODUCTION TO PASCAL, C.A.G. Webster, Heyden and Son, 1976.
No. of pages: 129. Price: £5.50, $11.

For several years now there has been an increasing need for an introductory text on programming which uses Pascal as the vehicle. Unfortunately, Webster has not given us that book. The following may indicate why.

In the preface the author claims coverage of an "essentially full version of the language, discussing where appropriate ... the original report and its revision (sic)". In fact the book describes the original (1971) language and supplements this with incomplete and inaccurate summaries of the 1972 revision. No warning is given that the language has developed vigorously since 1972. This makes the book almost useless in conjunction with compilers for the latest version of the language, Standard Pascal.

One might expect that a book on Pascal would pay some heed to modern ideas of programming methodology. Instead we find algorithms introduced by machine-language programs and illustrated by "spaghetti" flowcharts. The concept of stepwise refinement ("top-down programming") is not mentioned until three quarters of the way through the book and then only in the context of procedure declarations. No substantial guidance is given in vital areas such as program design, testing, debugging, correctness and maintenance. It might almost be a book on BASIC!

These global defects are compounded by a list of errors of fact, omission and commission which leaves a blemish in almost every page. The following are just a few of the more serious.

(a) Variable parameters of procedures are (wrongly) said to be passed by reference, in a section which manages to make the (very simple) parameter-passing rules of Pascal seem almost incomprehensible in their complexity.

(b) Several examples of bad practice in the use of real arithmetic are hardly compensated by a superficial warning about the comparison of real values.

(c) The operator NOT changes the state of the following operand, according to Chapter 4.

(d) Chapter 6, under the heading "Initializing variables", describes the definition of symbolic constants. The initialization of variables is also described, but without warning that the feature is not part of the defined language.

(e) There are many lexical, syntactic and logical errors in the programming examples, some of them seemingly calculated to cause the maximum confusion for the beginner. For example, despite a warning early-on about the precedence of relational operators in Pascal, almost all the more complex Boolean expressions in the book are wrongly bracketed (or not bracketed at all). The following, given as a way of skipping characters up to an asterisk or the end of file, is quite typical :

```
REPEAT s := input↑; get (input)
UNTIL  s:='*' AND  NOT eof (input)
```

In short, avoid this book.

W. Findlay
University of Glasgow

APPLICATIONS

(* Reports of applications come to the Newsletter from PUG members, primarily. If you know of applications which use PASCAL, please send us the details. *)

Progress Report on PLT - March, 1977

PLT (Programming Language for Teaching) is a machine independent CAI/CMI (Computer Aided Instruction/Computer Managed Instruction) system implemented entirely in PASCAL-6000. PLT features a concise structured lesson creation language implemented with a fast single pass compiler, an efficient interactive interpreter, and full lesson and student monitoring facilities.

The PLT system will automatically step individual students through a series of lessons and tests. Reports by student and/or lesson-tests can be generated using the system's reporting facilities.

PLT is in full production use at Lehigh University and is being used to implement a series of lessons on PASCAL programming.

PLT will be released as an unsupported product after completion of its system internals manual (about April 30, 1977). For further information please write to

Richard J. Cichelli — Christmas-Saucon Hall 14
Computer Science Group — Lehigh University
Department of Mathematics — Bethlehem, PA 18015

RUNOFF text formatter

A version of RUNOFF (the well-known text formatter available on the DEC-10 and other machines) is available in Pascal on the CDC Cyber 175. Educational institutions may get it on a free exchange basis provided you send a tape, expect no immediate response on bug-fixes, and do not distribute it to others. Documentation is also available. Write to Bob Foster, Computing Services Office, University of Illinois, Urbana, IL 61801.

STEP--A System for cross compilation.

Programs are written in Pascal. The target code is macro-generated with Stage 2 (using an intermediate code). Complete. Available for distribution. Michel Galinier (* PUG member *), Université P. Sabatier-Informatique, 118 Route de Narbonne, 31077 Toulouse Cedex, France. (* 1/5/77 *)

(* From a news brief in Electronics, March 17, 1977, p. 140 *): "Electro-Scientific Industries, Inc., Portland, Ore., is beginning to offer its own compiler for use by others on the [DEC PDP-11]. At the end of last year, Computer Automation Inc., Irvine, Calif., announced a combined compiler-interpreter for its own mini-computers. Both companies point out that Pascal . . . is simpler to use than either Fortran or Basic. ESI is aiming at applications in automated test and data-acquisition systems and in its own laser trimmers. Computer Automation likes it for developing compilers and translators."

(* T.E. Bell, D.C. Bixler, and M.E. Dyer, ("An Extendable Approach to Computer-Aided Software Requirements Engineering," in Structured Design, Infotech State of the Arts Conference, 10/18-20/76, pp. 3-27, also in IEEE Transactions on Software Engineering, 3:1 (January, 1977), 49-60) report that TRW used Pascal as the implementation language in its computer-aided system for maintaining and analyzing system software requirements.*): "The simulator generator transforms the [Abstract System Semantic Model (* a database *)] representation of the requirements into simulator code in the programming language PASCAL. The flow structure of each [Requirement Network (R_NET) (* the class of processing flow specifications *)] is used to develop a PASCAL procedure whose control flow implements that of the R_NET structure. Each processing step (ALPHA) of the R_NET becomes a call to a procedure consisting of the model or algorithm for the ALPHA. The models or algorithms are written in PASCAL." (* from p. 18 *)

PASCAL PRINTER PLOTTER

A listing (6 pages) of the Pascal code for the printer plotter described in PUGN No. 7 is available free. Write to Herb Rubenstein, University Computer Center, 227 Experimental Engineering, University of Minnesota, Minneapolis, MN 55455.

ARTICLES

DEVELOPMENT OF A PASCAL COMPILER FOR THE C.I.I. IRIS 50. A PARTIAL HISTORY.

Olivier LECARME
Université de Nice

1. ENVIRONMENT

The history which is the subject of the present paper takes place in the University of Nice, a medium-scale University with about fifteen thousand students. The department of computer science is very small, but delivers two different B.Sc. degrees in computer science (informatics), and has full graduate programs. About two hundred students attend these undergraduate and graduate courses.

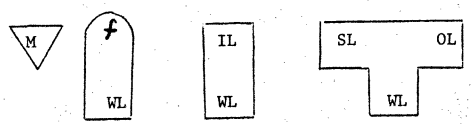
The University computing center serves the whole academic community with a C.I.I. Iris 50 computer, a medium-scale machine comparable in power and capabilities with an I.B.M. 360 model 40 or 44. The multi-programming system Siris 3 allows the execution of several jobs in fixed size partitions, the biggest possible size being 220 K bytes (stand-alone mode), and the most current sizes being 64K or 96K bytes. The department of computer science accesses this computer by remote batch processing, with a mean return time of about one hour.

Programming languages available are an assembly language (without macro definition capabilities), Fortran and Cobol. None of these languages may be considered an adequate support for teaching computer science, and especially for teaching programming to future specialists. Successive attempts to implement Pascal have consequently been done, with variable success. The main difficulties encountered are the lack of dedicated manpower, the weakness of software tools available, and the small amount of storage normally available on the computer.

2. FORMALISM FOR DESCRIBING TRANSLATORS

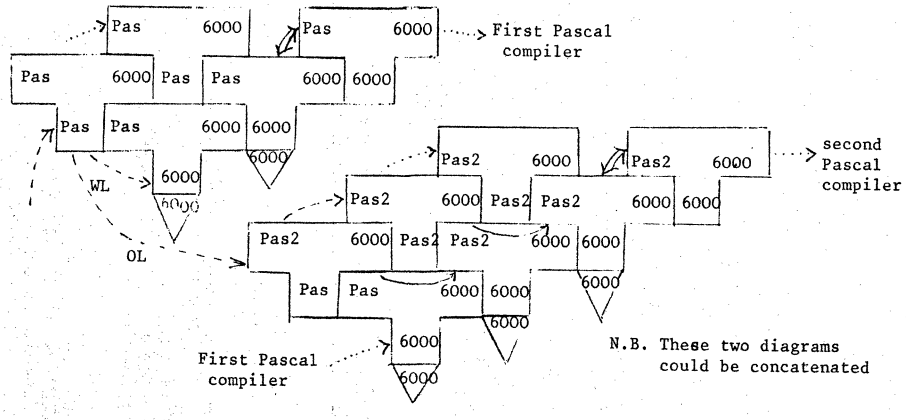
The so-called T-diagrams of Earley and Sturgis are very useful to describe the generation of translators by complicated bootstraps.

The four following different symbols describe the different programs :



The first one is a hardware processor, i.e. a computer or machine M. The second one is a software processor, i.e. a program realizing some unspecified function **f**, and written in the programming language WL. The last two are software processors realizing a specified function : the first one is an interpreter for language IL, and the second one is a translator from source language SL to object language OL.

By concatenation of different symbols, and provided that the same language always appears on both sides of any concerned frontier, we can describe the generation of a compiler by another one. We use four different arrows describing the translation processes : the solid arrow indicates that the target program is the translation of a source program by a programmed translator ; the dashed arrow indicates that the target program is produced by hand, either from scratch (no source program), or by modifying one of the languages concerned (this language is specified along the arrow) ; the dotted arrow indicates that the target program is a copy of another one, without modification ; the double-ended arrow indicates that two programs must be identical for validating the bootstrap. Using this formalism, we can describe the history of the two principal Pascal compilers for the CDC 6600 (the following diagrams abusively simplify the history) :



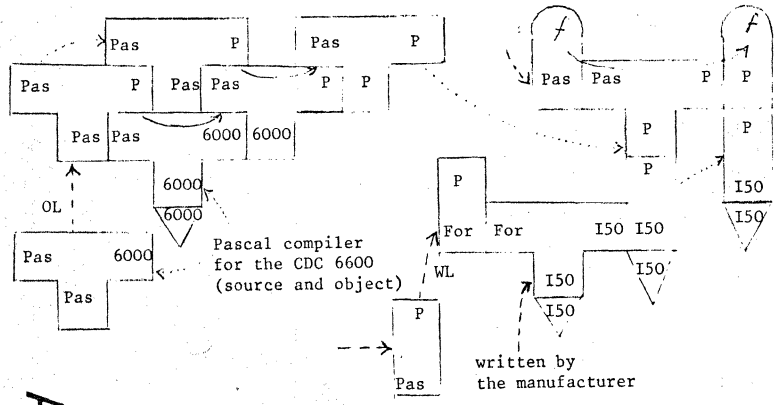
3. FIRST ATTEMPT WITH PASCAL-P

A translation into Fortran of the interpreter for P-code (first version of Pascal-P) had been made in Paris for a CDC 3600, and carefully written for being really portable. In fact, it was quickly implemented on the Iris 50, but it gave so disastrous performances in time and space that the compilation of a tiny program (by interpreting the compiler) would have necessitated exclusive use of the computer for about one hour. Explanations of this phenomenon are simple and interesting.

The packing and unpacking of P-code instructions fields were done with multiplications and divisions by powers of 2. The Fortran compiler did not recognize the special form of these operations, and generated ordinary code, which was especially complicated for integer arithmetic. This partly explains the slowness of the interpretation, other factors being the use of Fortran input-output routines, and the heavy overlay loading necessary because of memory problems.

These memory problems are partly due to the weaknesses of the first P-code, corrected in the last version : either the length of each data object must be explicitly indicated in its representation, or each object must be allowed the same storage size. This second solution had been chosen in Paris, because of better time performances, but it necessitated two memory words (32 bits) for each type Boolean, character, integer and real, because of the 58 bits necessary for sets when interpreting the compiler. The interpreter needs a "memory" of 24 K "words", i.e. in our case 192K bytes, plus the interpreter itself, the interpreted program and the Fortran execution support (principally input-output).

The following diagrams describe the two phases of the generation and use of a Pascal implementation with Pascal-P.



4. SECOND ATTEMPT WITH PASCAL-P

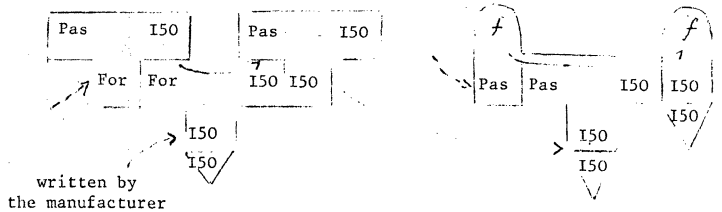
A completely new translation of the interpreter into Fortran was done, using the second version of Pascal-P. Some assembly language routines were used for packing and unpacking fields, manipulating length indicators for data, etc. All possible gains in time and size were carefully searched and programmed. The final version, used during the last academic year, permitted the compilation and execution of half-page programs at a tolerable cost, but no more. A student trying to have a four page program compiled would have exhausted all his available funds for one month ! This taught to students an extraordinary carefulness when trying their programs, and in many cases they managed to get a complete working program in only one run. This result is not so bad, but other frustrations were not tolerable, and this tool could not have been used for more than one year.

Moreover, the poor performances of this compiler made completely impossible to use it as a tool for developing an actual compiler, generating machine code : the compilation of the Pascal-P compiler would have necessitated more than 8 continuous hours of exclusive usage of the computer. This was, practically, absolutely impossible, especially within our local context.

The same diagrams in the preceding section describe this compiler.

5. A STUDENT JOB

During the same academic year, and using only his spare time, an undergraduate student wrote, partly in Fortran and partly in assembly language, an in-core load-and-go Pascal compiler. It happened, by incredibly good fortune, that this compiler was sufficiently well written to be usable by students. It is used during the present academic year, and gives a compilation cost of about 1.4% of that of the Pascal-P second compiler. It implements standard Pascal with only a few very minor restrictions, but prescribes very small limits on the size of programs, the number of identfiers, the complexity of all declarations, and so on. All in all, it is only a teaching tool for small programs, and it is completely impossible to make it usable for implementing a full compiler. However, it was a mean to wait for something better and more general, without too much frustration.



6. DEVELOPMENT OF THE TOOLS FOR THE FUTURE BOOTSTRAP

The compiler of section 3, 4 and 5 are only toy compilers ; they cannot compile themselves, and consequently they cannot be used for developing themselves. Since we have no other computer available in the vicinity, and no adequate software tool on our computer, many possible solutions had to be eliminated, especially those which use a powerful macro-generator. No funds were available for repetitive travels between Nice and other computing center, and still less for computer time in other centers.

The chosen solution is consequently very original, somewhat complicated to describe, but needing only a minimal amount of programming. Moreover, this programming is not done by us, but in another (wealthier) University, Université Paul Sabatier in Toulouse, France. Several processors are available in that later place : 1°) A Pascal compiler running on a CII Iris 80 computer, bootstrapped from a CDC 6000 computer by Didier THIBAULT, and described in Pascal Newsletter #7 ; it is a compiler from Pascal to Iris 80 machine language, written in Pascal and bootstrapped into Iris 80 machine language, according to the customary diagram :

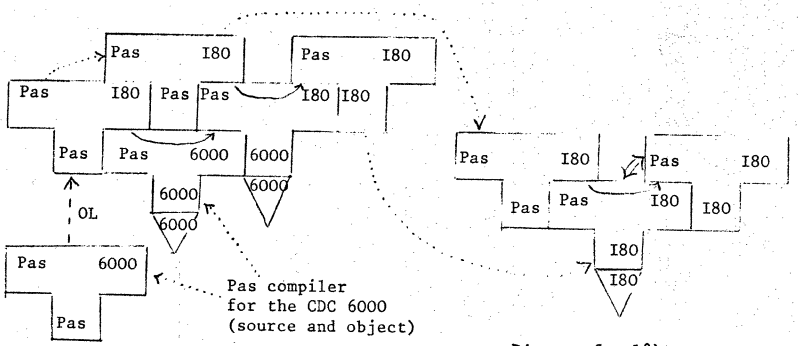


Diagram for 1°).

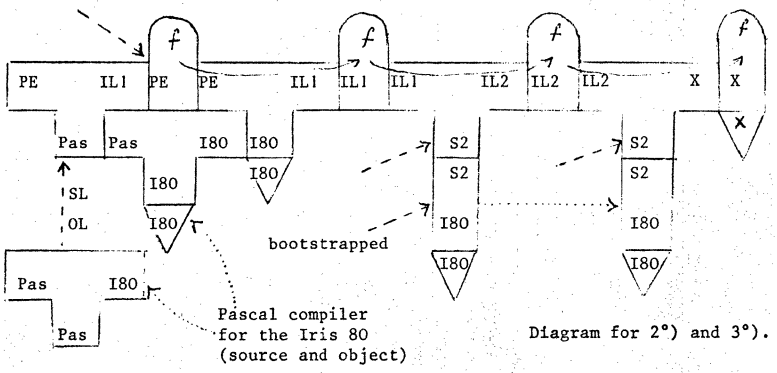


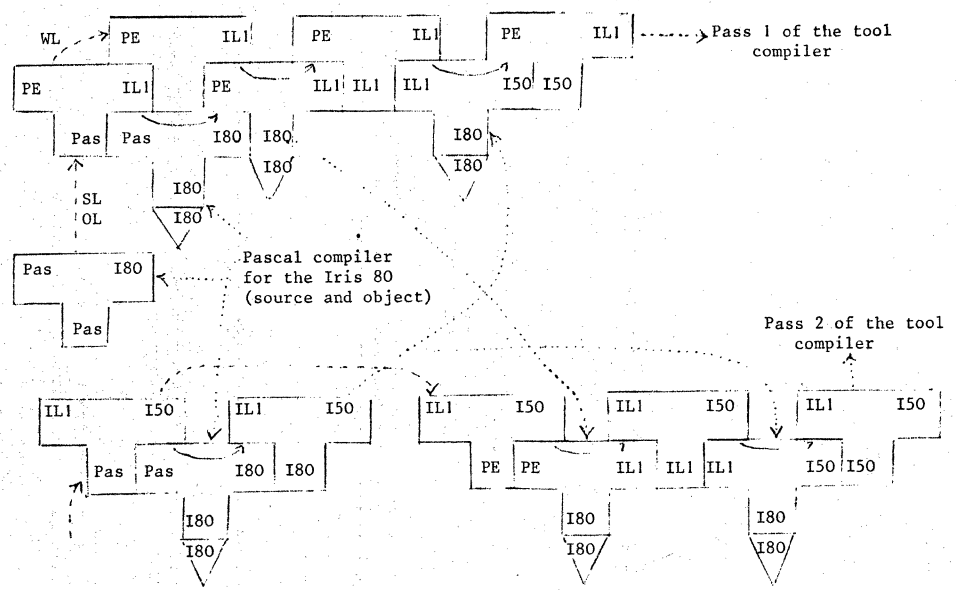
Diagram for 2°) and 3°).

2°) A compiler from a subset of Pascal (called Pascal-E) to an intermediate language IL1, written in Pascal. 3°) A translator from IL1 to a second intermediate language IL2, written in Stage 2; Stage 2 is itself an interpreter, implemented via a full bootstrap. IL1, IL2 and the corresponding compiler and translator were developed in Toulouse as a tool for cross-compiling, on the Iris 80, Pascal programs for several mini-computers.

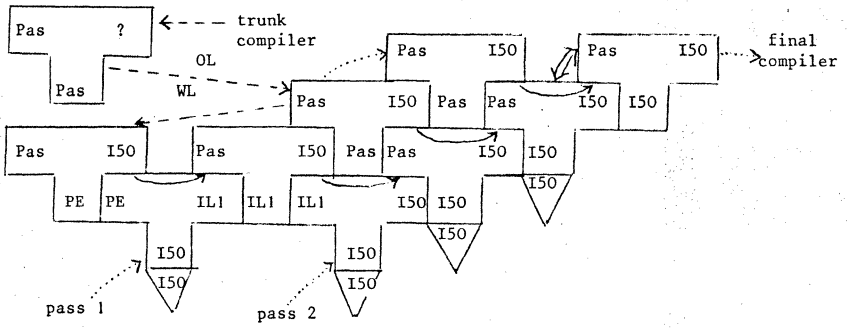
IL1 is a quadruplet language, as suggested by Gries and used between two passes in many compilers ; IL2 is a machine oriented language, especially designed for being easily translated into machine code for mini-computers, by a second translator written in Stage 2. The diagram above is for 2°) and 3°).

7. DEVELOPMENT OF THE TOOL AND FINAL COMPILERS

We wanted to avoid the bootstrap of a complete compiler, made by changing the object language on an existing one, because of the difficulties of debugging and tuning such a large program when using two computers distant by 600 kilometers. By writing by hand, in Pascal-E, a translator from IL1 to Iris 50 machine code, we can obtain the tool compiler we need, but in a two-pass form. This translator is very easy to write since IL1 is a quadruplet language. Some modifications are also necessary for the compiler from Pascal-E to IL1, to write it in the subset of Pascal it accepts, but these modifications are in fact trivial, since the only things to do are to replace, for example, writeln (code) by writelnc , or write (output,c) by write (c). By doing six translations on the Iris 80, according to the following diagram, we obtain a two-pass Pascal-E compiler usable on the Iris 50 :



The future and final compiler for the Iris 50 will be obtained by filling holes in the "trunk compiler" of H.H. Nägeli, as it was done with success in Tokyo for the Hitac 8000 ; it will be implemented with the tool compiler :



The two-pass tool compiler is intended to be usable in next March. It implements the minimal subset of Pascal needed for writing the compiler. Omitted features are reals, file declarations, most standard procedures or functions. Standard files necessary for implementing the compiler are built-in, an accessed via several pre-defined procedures. No other part of the language is omitted and packed structures are implemented. The translator from IL1 to Iris 50 produces code for the linkage editor, one module per procedure. This will permit segmentation and overlaying of the compiler, but without any means for checking the validity of access to global variables.

The final compiler will implement exactly standard Pascal, and produce modules for the linkage editor, as the preceding one. The whole process should be terminated at the end of 1977, and several computing centers are already interested in the final product.

(*Received 77/03/09.*)

A FURTHER DEFENCE OF FORMATTED INPUT

B. A. E. MEEKINGS

Computer Studies Department
University of Lancaster

In PUGN # 7, Barron and Mullins attempt to demolish the case for formatted input. Without wishing to blow up the controversy beyond reasonable proportion. I would like to add a voice in favour of formatting.

I feel that Barron and Mullins have rather missed the point, inasmuch as input data is unfortunately not always under our control; in addition, of course, it is unreasonable to exclude a feature from the language simply because it can be done in another (more long-winded) way. I have implemented formatted input in my Pascal P4 compiler forthe following, I think very good, reasons:

i. it is a simple modification to make, with relatively little overhead.

ii. it is entirely within the "spirit" of Pascal, making input consistent with output, which it currently is not

iii. it allows for reading in character strings to complement the similar output feature

iv. I am working in general area of simulation languages, and hence attempting to woo users away from the traditionally Fortran based languages - I see formatted input as one less obstacle for them to overcome in the transition.

I am not in any sense an advocate of Fortran, but I do feel that the association between formatted input and Fortran is no valid reason for its exclusion from other languages.

In short, the addition of formatted input, to supplement the existing unformatted input facilities, can only enhance an already versatile language.

(*Received 77/03/21.*)
(* Meekings is not a PUG member yet.*)

PROPOSALS FOR PASCAL

George H. Richmond
University of Colorado
Computing Center

THE REPRESENTATION OF PASCAL FOR COMPUTER INPUT

The original lexical definition of Pascal was closely tied to the CDC character set. The current implementation allows for complete representation of all Pascal elements in the ASCII character set except the up arrow which is used for pointer and file references. In this case, circumflex is the ASCII character that is used.

This lexical representation should have a 48 character alternative for computer systems with restricted character sets. Some obvious equivalences are period-period (..) for colon (:) (this is almost always true for the CDC implementation), period-comma (.,) for semicolon (;), and period-equal (.=) for replacement (:=). Additionally, two letter alternatives for relational operators should be allowed. Brackets for subscripts could be (. and .) or (/ and /).

Whether or not to always accept the 48 character representations is an open question.

COMPILE OPTIONS

The Report [1] does not mention compiler options enclosed in comment symbols, but perhaps this means of defining compiler options should be formalized. Several compile time options like listing control, code generation, and source line width should be universally defined.

INTERNAL CHARACTER SET

Pascal could be made the first language to standardize the ordering of characters for the basic data type CHAR. This standard could be ASCII. Thus the basic data type CHAR will have 128 elements. At the moment, the CDC implementation is stuck with the anachronism of a character set based on a 6-bit element. It would be reasonable for text files to be mapped between the internal ASCII set and the external operating system character set. The normal character set for a particular machine could be accessed without translation by using a packed file of the appropriate integer subrange type.

An alternative solution to the problem of antiquated character sets would be to provide several CHAR types. ASCII could be a keyword which defines the 7-bit ASCII character set. EBCDIC would define that 8-bit set. The local machine implementation of characters would be CHAR. There should be character set conversions across assignment statements.

REMOVAL OF CURRENT RESTRICTIONS AND ASYMMETRIES

The restrictions and asymmetries outlined below are made with reference to the CDC implementation of Pascal.

First, and foremost, the designation PACKED should tell the compiler to optimize storage usage instead of speed of access in data structures. It should not have any other effect upon constructs in the language. Unfortunately for the CDC implementation, this is not true. One cannot compare or output unpacked arrays of characters, but this can be done for packed arrays of characters. This particular asymmetry is reminiscent of Fortran in its arbitrariness.

There is an implementation problem in passing elements of packed structures as VAR parameters which will probably have to remain.

A bothersome restriction is set size. Sets should be allowed to have any size, not just some convenient but fixed machine size. It is difficult to justify the exclusion of the last 4 elements of the CDC character set just because there were only 60 bits in the CDC word.

If a subrange declaraction of INTEGER exceeds the normal precision of the usual representation, an automatic extension to multiple precision arithmetic should occur. There should be some way to declare the minimal precision required for the REAL type so that multiple precision arithmetic could be used if necessary. In this manner, a precision sensitive algorithm could be run on different precision machines with good results.

FUNCTIONs should be able to return any type. Identifiers should be unique to their entire length.

THE PROGRAM DECLARATION

Aside from specifying the name of the main program, the program declaration contains a list of file names. The current usage of the declaration in the CDC implementation is to allow immediate opening of all files upon entry into the main program and to establisn the ordering of files for the positional substitution of system file names that is possible in CDC operating systems. The first action is unnecessary. The second action should be clarified in the Report [1] or regarded as a CDC implementation feature. Neither INPUT or OUTPUT should be mandatory on the program declaration. All files must be opened explicitly before test or data transfer. Otherwise, an error is diagnosed. Close operations should also be available.

VARIANT RECORDS

The current definition of variant records is quite useful and has been cleverly utilized to subvert type checking within the CDC implementation of the language. This is unaesthetic even though it is necessary. There ought to be a better way.

Unfortunately, the current definition of the language does not allow the tag field of variant records to be automatically set when a variant record is allocated or a variant field is stored and to be tested for correct type when a field is fetched. This checking should be done to protect the run-time system from the lazy or careless user. Perhaps another formal compiler option should be defined to disable this type of protective code.

THE CASE STATEMENT

A decision in a case statement may be implemented by a jump table or series of tests. The compiler should choose which technique to use based on the type of expression involved. Perhaps a type identifier in addition to the normal expression would help narrow the range of values and allow the faster jump table to be selected. In any case, an ELSE exit is highly desirable. It is a waste of time to force the programmer to protect each case statement with an if statement. Also, it would then be possible to make case statement tests on strings, large integers, or real numbers. Another extension would be to allow the subrange notation for case labels so that ranges of values could be directed to a statement.

BOOLEAN EXPRESSIONS

Boolean expressions should be computed only as far as necessary to establish the value of each subexpression. AND is FALSE when the left operand is FALSE. Then the right operand could be ignored. Similarly, OR is TRUE when the left operand is TRUE. The ultimate value of the entire expression would still be correct by doing this partial evaluation, and the expression of loop termination conditions when indices go out of bounds will be much simpler.

CONSTANTS, DECLARATIONS, AND CONSTRUCTORS

The Pascal language needs a means of constructing structured constants. In fact, Wirth [2] has defined constructors for this

purpose. It should be implemented.

In constant declaractions, it should be possible to perform compile time computations using constants and previously defined constant identifiers.

VALUE INITIALIZATION AND OWN VARIABLES

The current Pascal has a large core requirement because it does not have value initialization and it is not overlaid. Value initialization of structured variables can be done using the constructors mentioned above.

Value initialization should be possible in procedures other than the main program. These variables would be initialized on each procedure entry. On most machines, this will require run time code for initialization instead of loader initialization.

Own variables (in the sense of Algol 60) should be allowed, and would be initialized just once at load time.

PROCEDURE AND FUNCTION TYPES FOR COMPILE TIME CHECKING

One omission in the definition of Pascal in the usual strict compile time type checking is the unchecked correspondence between declaraction and usage of procedures and functions passed as parameters to other procedures and functions. This omission opens the run time system to mysterious collapse when procedures are incorrectly called. This compile time check can be done in one pass compilation if procedure and function type identifiers can be defined. The type would have the attributes of denoting a procedure or function, the number of parameters, and the type and VAR property of each parameter or result. This would actually simplify the syntax of a parameter list by eliminating the need for the keywords FUNCTION and PROCEDURE. If the parameter position is typed by a procedure type identifier, then the actual name of a procedure must be passed at call.

DYNAMIC ARRAY PARAMETERS

Although some limited means of passing variable sized arrays is desperately needed in Pascal, Jacobi's proposal [3] is too limited in scope. A dynamic array parameter should be indicated in a parameter list by the inclusion of the keyword DYNAMIC before the type identifier. Any actual parameter which conforms to the type, except for array bounds, would be accepted. This allows for arbitrary packed structures. The prohibition against other than element wise access should only apply down to the last array substructure.

NEW BASIC TYPES AND OPERATORS

A major advantage of Pascal over other programming languages is its expressive power in data structures. Because more information about the data being operated on is available to the compiler, better code can be generated to handle the manipulations. For this reason, the basic types of Pascal should be expanded. The COMPLEX data type is one that should be added.

For similar reasons, the exponentiation operator should be added to the language.

Another extension I propose requires more justification because of its impact on the implementation. STRING should be added as a basic type along with operators, standard procedures, and functions for concatenation, extraction, pattern matching, and type conversion. The closest approach available now is a record composed of an integer character count and an array of characters. This is an inadequate alternative as the compiler cannot easily recognize this as a string and the programmer is burdened with providing a plethora of auxiliary routines. The resulting code is less efficient than what is possible if the type STRING was defined.

Of course, well defined automatic coercions (in the sense of Algol 68) must be available between strings and arrays of characters. Additional standard procedures and functions equivalent to tne CDC Fortran ENCODE and DECODE routines should be available. when possible, the compiler should revert to the older pattern of fixed sized array of characters instead of treating all character string constants as STRINGs.

TRANSFER FUNCTIONS

Transfer functions between scalar types and their character string names should be available. There should be a type check defeating function which regards its source and destination as bit fields of some appropriate width. This function would eliminate the need for tne variant record subversion. Inverse functions for ORD across all basic types might be considered to be the type check defeating mechanism.

EXTENSION OF RELATIONAL OPERATORS TO STRUCTURED TYPES

Relational operators already extend to structured types in the one case of packed array of character. They should extend in tnis manner to all structured types. To do so there must an ordering of elements within a structured type from first to last and the comparison must takes place in this order. This straightening should apply to several other areas of the language as in input/output and constant formation.

FILES AND TEXT FILES

The Report [1] allows attaching the keyword PACKED to file types but the CDC implementation does nothing with it. Actually, there is a confusion in this area of file types. There are really three types of files. There are unpacked files, packed files, and text files. The type FILE OF CHAR, PACKED FILE OF CHAR, and TEXT are not equivalent.

In particular, an unpacked file of some type aligns items of the type on any particular machine word (or byte) boundary that is convenient and provides quick access. A packed file of type is implemented with every reasonable effort to not waste one bit of disk or memory space. Specifically, on the CDC machine, FILE OF BOOLEAN would be stored with one boolean value per 60 bit word and PACKED FILE OF BOOLEAN would be stored with 60 boolean values in one word. Also, with packed file of subrange of integer type, it should be possible to access any packing of data on disk independent of word boundaries. The only operations available on packed or unpacked file types are GET and PUT (or tne shorthand READ and WRITE with no type conversion) along with assorted status testing and positioning operations.

The files of type TEXT are fundamentally different from the other two file types. First, the procedures READ and WRITE are available with their full formating and type conversion possibilities. But a text file is not a FILE OF CHAR. It is a specially handled character file witn lines of text. It has line boundaries which a FILE OF CHAR does not have. In fact, each line of text should be treated like a value of type STRING.

Also, text files come in two varieties, paged and unpaged. This attribute is established by declaration at compile or open time. An unpaged TEXT file would be associated with devices such as card reader, card punch, magnetic tape, or teletype input. A paged TEXT file would be associated with a line printer or teletype output. TEXT files must operate with the correct order of input and output on interactive devices. It may be necessary to declare files as being interactive in order to keep the run time system straightforward.

The user should not be responsible for placing carriage control in column 1 of every line of paged output. The paged output routine should normally provide a blank for the line printer but omit it for teletypes. A call to the PAGE procedure should set up carriage control (like page eject or form feed) as needed.

Text files are subject to translation between the operating system character set and the internal character representation. The rules for skipping from one line to the next have not yet been well formed and will have to account for the straightening process of structured types. The problem of reading blanks before end of file should be resolved once and for all. It should be possible to read one line of text into one STRING type variable and perform type conversion later.

For paged text files, it should be possible to automatically invoke user supplied procedures at top and bottom of forms. Other user supplied procedures could be invoked on various fault conditions for all file types.

FORMATTED INPUT AND OUTPUT

It is not necessary to resurrect the Fortran format to handle text file formating in Pascal. The WRITE procedure field width specifications are fine. They should be extended to the READ procedure. It should be possible to read and write delimited strings of characters. There should be an option for separator characters other than blank between input or output items.

FILE HANDLING

Text files should be processed strictly sequentially. Random positioning should be allowed on non-text files. Since most operating systems provide for file structures that are more complex than currently defined in Pascal, there should be some generally agreed upon extensions to file operations that are not mandatory. The CDC implementation does have the extensions of SEGMENTED files. The CDC version needs additional extension for multiple file files. For example, add GETEOF, PUTEOF, and WHILE NOT EOI (for End-Of-Information).

The current CDC implementation does a rather poor job of file positioning at open and close time. Explicit file open and close operations are necessary. A rewind or no rewind option is vital for both. Other file attributes like system file name, buffer size, procedures for handling data exceptions should have reasonable defaults but be open to user specification.

OVERLAYS

The Pascal language needs overlays. The first use would be to reduce the size of the compiler by doing value initialization functions in one overlay and the main compilation process in another. A halfway overlay attempt already exists in the CDC implementation to issue compiler error messages.

Designation of overlays can be achieved by compile time options in comments or by adding the keyword OVERLAY to the syntax. The choice of which to use is open and should be decided. Overlays are organized by procedure or groups of procedures. Explicit overlay calls should not be necessary as in CDC Fortran. The compiler can recognize a call to a different overlay and generate the appropriate code.

A good proposal for an overlay mechanism has already been made [3]. However, it already exceeds the capabilities of the CDC operating system. To accomodate that system, no more than two levels of overlays could be allowed and the implementation would be even easier if overlaid procedures could be called only from the outer block.

As stated in the overlay proposal, overlays can be viewed as is the designation PACKED. A particular Pascal implementation will try to follow the overlay directives and the program will always run correctly. However, the object code may not be as deeply overlaid as specified.

PREAMBLES AND POSTAMBLE

A compiler does not stand by itself within a computer system. A well developed language system must have a wide range of subprograms available for use. One reason that Fortran will be hard to displace is the large number of subprograms already developed for it.

The implementation of separately compiled procedures in CDC Pascal was a gigantic step forward in increasing the usability of the language. But now the user is burdened with declaring all external procedures he intends to use. The declaration is necessary but it should come from the language system rather than the user.

The compiler cannot be reassembled every time a new subprogram is added to the library, and it should not carry declarations for every possible external subprogram when only a small number of them for a specific application will be accessed.

The solution is to allow the selection of several preambles which initialize the compiler to a particular application environment. The compiler would look to the preamble for each declaraction section (PROGRAM, LABEL, CONST, TYPE, VAR, and subprograms) first and then compile the corresponding user declaration section. Preambles should be input as ordinary text or specially processed system text records.

A provision for a postamble would be useful to allow driver main programs in a student environment or for a non-code producing dummy main program when compiling library subprograms.

The preambles and postamble allow a user job to be compiled in any desired environment. By allowing full procedure parameter description in the preambles, including procedures passed as parameters, complete compile time checking of all external subprogram linkages can be obtained.

Also, some mechanism of protecting access to the elements of a structured type introduced in the preamble is desirable. This would be useful in making certain data structures appear as basic types to the user.

[1] Jensen, Wirth, "Pascal User Manual And Report", 2nd Edition, Springer-Verlag, 1975.
[2] Wirth, "Algorithms + Data Structures = Programs", Printice-Hall, 1975.
[3] Pascal User's Group, "Pascal Newsletter", No. 5, September 1976.

(*Received 77/03/24.*)

A PROPOSAL FOR INCREASED SECURITY IN THE USE OF VARIANT RECORDS

WILLIAM BARABASH
CHARLES R. HILL
RICHARD B. KIEBURTZ
SUNY AT STONY BROOK
STONY BROOK, NEW YORK 11794

The use of variant records in most Pascal implementations is dangerous because most compilers do not emit a check for conformity with the value of the tagfield when a variant field is referenced. Indeed, the latest version of the Revised Pascal Report defines a language in which the tagfield may even be absent, making conformity checks impossible! Even so, when the tagfield is present and the compiler does emit conformity checks automatically, the programmer still has the ability to dynamically assign values to the tagfield.

We propose that the variant field of a record be protected from such abuse, either accidentally or intentionally. This means that the compiler should be required to emit conformity checks when a variant field is accessed; that the tagfield must always be present in every variant record; and that the programmer not be allowed to alter the tagfield in a variant record by means of a simple assignment statement.

Currently, a variant record can be created dynamically when the standard procedure New is applied to a pointer variable that is bound to a variant record type. This standard procedure has the ability to initialize tag fields to constants specified in the call. We propose that thereafter the type of the variant record is frozen by the values of the tagfields. The fields within the record can all be referred to; however, if a field in the variant part of the record is referred to, the tagfield will automatically be tested for conformity.

This is not sufficient, because variant records in a Pascal program can reside in the stack, being created on block entry. Such records can only be initialized to "undefined". Also, during the lifetime of a dynamically created variant record, it may be created and used, then put on a free list, then used subsequently. The subsequent user might want a different set of values assigned to the tagfields of the record. To get around these difficulties, we propose a new standard procedure which will

1) set all of the fields in the record to "undefined", then
2) initialize the tagfields in the record to the constant values specified in the call.

A call to this procedure would be exactly like a call to standard procedure New, except that the first parameter would designate an already--existing record variable instead of a pointer variable. Such a procedure might be called "Renew". Note that the use of Renew has one chief drawback, namely that when a variant record is created, space must be allocated for the largest possible variant field. On the other hand, if a variant record is created by means other than the standard procedure New, the maximum space must be allocated anyway. Furthermore, garbage collection would be simplified: there would be no need to provide more than one parameter to standard procedure Dispose.

Lastly, it might be argued that enforced run-time conformity checks when a variant field is frequently referred to can severely degrade the performance of a Pascal program. We propose a slightly modified with statement which can open the scope of a variant record with a tagfield value assertion. The assertion is checked at run-time once every time the with statement is entered. Within the body of the with statement, any reference to a variant field of the record can be checked for conformity with the asserted values of the tagfields at compile time. Such a statement would have the syntax

with recordvariable (const 1,..., const N) do S

meaning that we assert that the variable "recordvariable" has tagfields whose values are "const 1" ,..., and "const N", as if the call

Renew (recordvariable, const 1,..., const N)

was made to initialize the record.

(*Received 77/03/27.*)

Update on UCSD PASCAL Activities

Kenneth L. Bowles
Institute for Information Systems
University of California San Diego
La Jolla, California 92093
(714)452-4526

17 April, 1977

LSI-11 Software

UCSD has recently started using a single user software system for microcomputers, with all major programs written in PASCAL. The compiler is based on the P-2 portable compiler distributed by the ETH group at Zurich, but it generates compressed pseudo-code for a much revised P-machine interpreter. As currently implemented on the LSI-11, compile speed is about 700 lines per minute (1000 on the PDP11/10). The system includes an interactive monitor, editor, utility file handler, and debugging package in addition to the compiler and interpreter. With 56K bytes of main memory, and dual floppy disk drives, it has proven more convenient and faster to do all software development on the microcomputer than to cross compile from a big machine. Whereas we have been using versions of this system that depend on I/O support from the RT11 operating system distributed by Digital Equipment Corp., our new system is independent of any external software support. The resident monitor, in terpreter, and run-time support package occupy an aggregate of about 10K bytes of memory.

Operation of large programs is facilitated through the concept of "Segment Procedures", which are rolled into memory only while actually invoked. The compiler (20K bytes), editor, and file handler are all separate segment procedures. One segment procedure can call others, and segment procedures may be declared nested within other segment procedures, to allow flexibility in memory management. The user's data space expands (or contracts if necessary) to take advantage of as much memory as possible after the appropriate code segments have been loaded.

Our plan is to have the new system completed to the point where it may be released to others by mid summer, 1977, with documentation package included. During the summer, we also plan to complete a graphics support package (including an editor for graphics oriented CAI materials), an assembler for PDP11 native code, and a compiler option allowing selected PASCAL procedures to generate native code rather than P-machine pseudo code. The system is designed to make relatively painless the problem of adding native code routines programmed in assembly language, allowing a user to augment the set of built-in functions and procedures where efficiency is important. This note has been composed and printed using a proprietary extended version of the text editor intended for use with a CRT display, which should be ready for release by late summer. The system should be usable on any PDP11 system capable of bootstrap loading from RX11-compatible floppy disk drives, or from the drives supplied with the Terak Corporation LSI-11 based machines (see next section). Further details may be obtained, on request to the address given in the heading, in separate notes titled "Status of UCSD PASCAL Project", and "Preliminary Description of UCSD PASCAL Software System".

LSI-11 Hardware

In addition to the well advertised PDP11/03 systems available from Digital Equipment, several smaller companies are offering stand-alone computers based on the LSI-11 that would be directly suitable for our software. We have been particularly interested in using a stand-alone machine with low cost graphic display for interactive educational applications. In connection with the EDUCOM Discount Program (see EDUCOM Bulletin, Spring, 1977), it now appears virtually certain that the Terak Corporation 8510A will be available to member institutions for about $5300 per machine (LSI-11, 56K bytes RAM, single floppy disk, CRT for superimposed but independent text and graphics, keyboard, RS232 asynchronous interface for network or printer connection). An example of the graphic display of this machine is attached to this note.

Other Microcomputers

Anyone who attended the West Coast Computer Faire in San Francisco should have come away impressed that small stand-alone microcomputers are big business and here to stay. It is possible to re-implement our PASCAL based software system on systems based on any of the most popular microprocessors within about 3 months of work by one programmer. At UCSD we have started to re-implement for the Zilog Z80 OEM series of modules, which could serve as the basis for PASCAL interpretive operation roughly as fast as the LSI-11. At the Faire, we talked with principal officers of most of the well known microcomputer manufacturers who sell to the hobbyist market, and encountered almost uniform enthusiasm for the idea of making PASCAL available on an industry-wide basis. On the basis of those conversations, there is a reasonable chance that our PASCAL system will be available later this year for use with the 8080A, 6502, and M6800 microprocessors in addition to the LSI-11 and Z80.

Proposal for Manufacturer Independent PASCAL System

There is widespread frustration, among those who make and sell microcomputer systems, that only BASIC is generally available, and that no two BASIC implementations are alike. Many of those we talked with at the Faire asked whether PASCAL would be standardized, to avoid the problems they encounter with non-standard BASIC (in addition to providing a more powerful programming vehicle). Even a casual reading of the PASCAL User Group newsletter is enough to convince one that: a) people are finding it necessary to enhance PASCAL for their own particular applications; b) the heterogeneity of the enhancements already reported is so great that no committee exercise is likely to produce a standard.

As an alternative, we believe that a chance exists to establish a defacto standard for PASCAL, at least for small systems, by starting a bandwagon effect in the microcomputer industry. A good definition of the underlying language for such a standard is contained in the Jensen and Wirth "PASCAL User Manual and Report". To implement a complete interactive software system, with adequate efficiency to run on a microcomputer, we have found it necessary to add built-in functions and procedures for handling text and graphics, and an EXIT(<procedurename>) built-in for clean termination of highly recursive programs. We have implemented SETs of up to 255 members in a way that uses memory efficiently, as well as Packed Arrays of BOOLEAN. For READ from a keyboard, the implied GET has to happen before the implied transfer from the window variable associated with the file. For handling floppy disks and other small storage media, we use the DEC standard of 512 byte blocks, and allow logical records conforming to any structured type allowed in PASCAL. In most other respects, we have been able to conform closely to the language defined in the Jensen/Wirth book.

If one common PASCAL based software system were to become available almost simultaneously for most of the mass distribution microcomputers, that system would establish the basis of a defacto standard for small stand-alone computers. Changes to such a system would certainly be needed with experience, but those changes might well be made readily available to most users through "down line loading" of object code through the dialed telephone network. Control of the PASCAL language standard might well be vested, at least temporarily, in a committee appointed by the PASCAL User's Group. Fast turnaround communications among the members of such a committee could be supported by "Electronic Mail" techniques over the dialed telephone network. The verbal responses we received from the manufacturers at the Computer Faire suggest that an unusual opportunity, that may not be repeated, exists in mid 1977 to establish a defacto standard in the manner described here. We invite the PASCAL User's Group to join with us at UCSD in bringing this about this summer. In most respects, the language and system definition design questions can be separated from implementation details. We have sought support to allow some of the advanced computer science students at UCSD to perform the implementation work on as many of the microcomputers as possible. Representatives of other institutions would be welcome to work with us in La Jolla, either on system definition or implementation. However we will not be able, ourselves, to devote a major percentage of our working time on definition of a standard.

Interested readers are invited to request copies of the following separate notes pertaining to the points discussed in this section: "An Appeal for Support of Manufacturer Independent Software", "Direct-Dialed Tele-Mail", "Proposal for EDUCOM Software/Courseware Exchange", "Minimum Cost Tele-Mail", "Student Projects for UCSD PASCAL System", "The Quest for a Cheap General Purpose Stand-Alone Computer".

Introductory Textbook

For the last two years we have used PASCAL as the basis of the large attendance introductory course in problem solving and programming at UCSD. The course is based on a textbook by this writer, that so far has been printed in the campus print shop. Student responses have been unusually favorable, and the course reaches more than two-thirds of the undergraduate population even though it is treated as elective for most majors. This response results partly from the non-numerical approach of the book, partly from student interest in our interactive system on the PDP11's, and partly from our use of Keller's Personalized System of Instruction (PSI) as a teaching method. Though suitable for PSI, the book can also be used as the basis for a conventional course. At the invitation of Professor David Gries, acting as computer science area editor for Springer Verlag publishers, the book will be published in paperback form this summer. The production schedule will be tight, and we anticipate that the first copies will be available barely in time for the start of fall quarter classes in late September. Springer is interested in knowing who might be interested in using the book and when. Unfortunately, alterations to make the non-numerical approach more readily accessible on many machines will make it difficult to circulate advance copies of the final text until late June at the earliest. We will be happy to forward inquiries to Springer.

Though very popular with the students, the non-numerical approach of the book has been difficult to sell to most of the publishers. The approach used so-far has depended upon programming examples using English text, and requires STRING variables and supporting built-in functions that we have added to PASCAL. In spite of this, the students learn the same programming skills that are taught in courses using traditional algebraic problem examples.

Since the inception of our project, we have wanted to orient the course to teaching with graphics oriented problem examples, using an approach motivated by the "Turtle Graphics" used by Seymour Papert of MIT. The microcomputers now becoming available make it possible to teach with a graphics orientation at virtually no higher price than needed for non-graphic materials. Accordingly, the textbook will be revised to augment, and often replace, the text oriented examples with graphics examples. For potential users lacking a microcomputer with graphics display, several alternate possibilities exist. Our built-in functions and procedures for graphics should be relatively easy to add to existing PASCAL compilers for other machines, and we will supply documentation to assist in that process. A description of the built-in's needed is contained in the note "Status of UCSD PASCAL Project" already cited. The implementation will assume a graphic display based on the "bit-map" principle, for which many devices are available in the microcomputer industry. Alternate display drivers will also be provided for the Tektronix 4006, 4010, ... series of direct-view storage tube terminals. Successful, though crude, plotting of the graphic output will also be possible on ordinary line printers. High quality graphic output is possible on matrix printers such as those made by Printronix (the graphic example attached to this note), Gould, Varian, and Versatec.

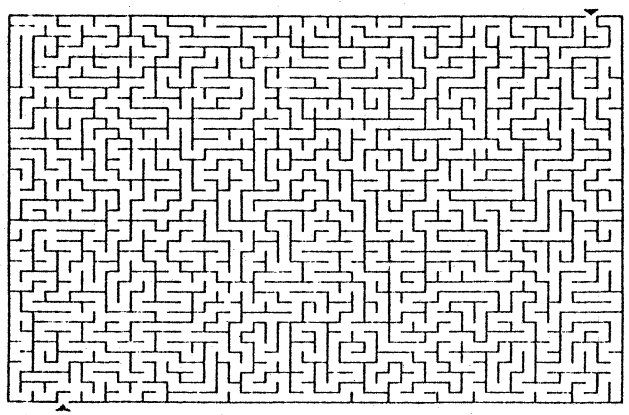
B6700 PASCAL Compiler

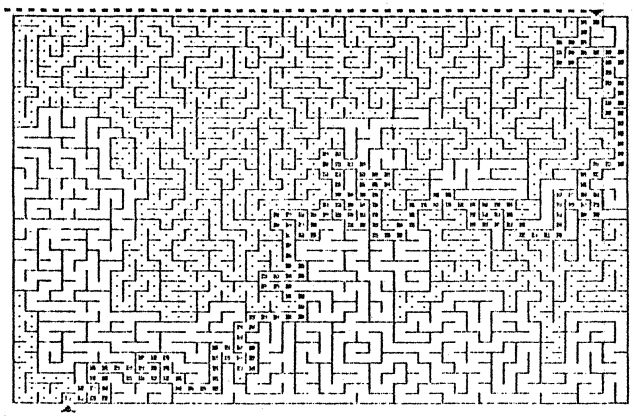
A PASCAL compiler which generates native code for the B6700 is now in operation at UCSD and available for distribution from the UCSD Computer Center. The compiler is written in PASCAL, and is based on the same variant of the P-2 portable compiler on which we have based the microcomputer implementation. Compile speed is about 5000 lines per minute of logged processor time. This compiler has been used for teaching large classes at UCSD for the last two months. As far as we know, most of the serious bugs in the original P-2 compiler have been corrected in both the B6700 and microcomputer implementations. The B6700 compiler provides access to most of the extensive file handling features of the B6700. At present, no implementation documentation has been completed for the B6700 compiler. The Computer Center will almost certainly generate such a document given an indication of interest in using this compiler by other institutions. Readers interested in obtaining a copy of the B6700 compiler should contact Henry Fischer, UCSD Computer Center, La Jolla, CA 92093 (714)452-4050.

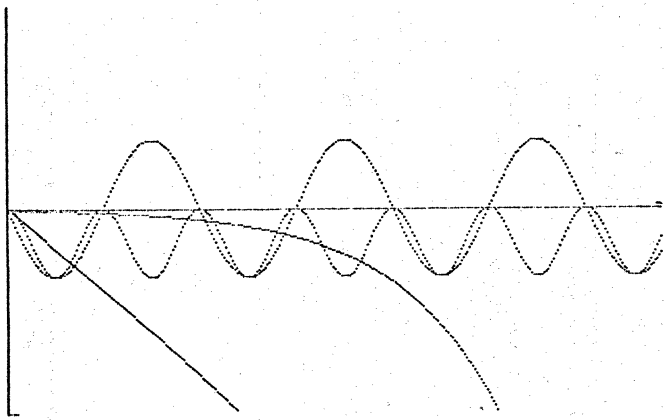
Apology to Correspondents

I offer an apology to the many people interested in our PASCAL work who have tried unsuccessfully to reach me by telephone or letter in the last few months. Currently I must depend upon several pooled secretaries who are not easily accessible. Having been occupied with a heavy teaching schedule, and with a committee assignment consuming one or two full working days per week, the correspondence has piled up. The series of titled notes and position papers cited earlier have been generated in self defense as a way to answer the many inquiries. The committee assignment has entered a dormant period. Future written requests for these papers will be answered promptly, but telephone inquiries may remain difficult until the re-write of the book is completed.

(*Received 77/04/20.*)







These figures drawn by small PASCAL programs on Terak Corp 8510A at UCSD

Contact: K L Bowles
(714) 452-4526

SOME COMMENTS ON PASCAL I/O

While admitting that PASCAL has I/O specifications involving the concept of files and the GET and PUT statements that are consistent with the flavour of the language and with theoretical manipulation of data, I feel that it is lacking in simple, easy to use I/O and in flexible I/O.

In any practical programming application, I/O is used for two main functions:

(a) Input of data from, and output of results to the real world.

(b) Permanent storage of data external to the program but internal to the computer, e.g. on tapes or disk.

Concerning the first function, I feel that, not withstanding the READ function in PASCAL, the use of TEXT files can be rather cumbersome and tedious. This is particularly so when dealing with string input (what delimits the string?) and when being used by a beginning programmer. I would like to see some form of simple I/O akin to the free format I/O of the PL/I GET LIST and PUT LIST concepts.

I have less of a complaint concerning the second function, but would suggest that information to be stored is often not homogenous as is effectively required by PASCAL files. One could argue that different types of data should be stored in different files, but this raises the problem of correlating the data in the files. Alternatively, one could use a file based on a RECORD type with a variant part, but this implies a varying size to the logical units of the file and may be difficult or cumbersome to implement on some computers. Finally it would be nice to be able to easily randomly access files and to update existing files in place.

I have not yet sufficiently formalised any alternative or additional I/O specifications for PASCAL and would be interested in hearing from anyone with ideas along these lines. Note that I consider it essential that any such specifications should as far as possible follow the PASCAL principle of being machine independant.

Chris Bishop
Computing Centre
University of Otago
P.O. Box 56,
Dunedin
NEW ZEALAND.

(*Received 77/04/07.*)

McMASTER UNIVERSITY
HAMILTON, ONTARIO, CANADA
L8S 4K1
DEPARTMENT OF APPLIED MATHEMATICS

Mr. Andrew B. Mickel,
Editor, Pascal Newsletter,
Computer Center,
University of Minnesota,
Minneapolis, Minnesota. 55455
U.S.A.

January 14, 1977.

Dear Andy:

From the correspondence on "standardization" in PNEWS 5 and 6 it seems fairly obvious that Humpty Dumpty's meaning of the term meaning is the rule. The letters seem to fall into two categories: on the one hand we have calls for the formation of an "official" standards committee, and on the other claims that "standard Pascal" is being adhered to, but necessary modifications need to be made for a variety of reasons. I put myself into the second group.

What are the objectives of standardization?

If the objectives are to ensure that a program written in establishment A can be run at establishment B without any changes whatsoever, whether or not A and B have exactly the same computers operating under exactly the same operating system, then I maintain that this is jest a pipe dream, because even a program written in ANSI standard FORTRAN in an IBM shop will need to be worked over if it is to run in a CDC shop. Moreover, a small change in an operating system can entail changes in the implementation of a language even if the change is completely transparent to the users of all other languages in the same establishment.

What I am driving at is that only in the case of an operating-system-independent language, not just a hardware-independent language is there any hope of us being able to achieve "perfect" standardization.

Pascal happens to be less of an OS-independent language than, say, FORTRAN. Eighteen months ago it took me quite a bit of effort to make some implementation-type changes when McMaster went from SCOPE 3.4.3 to 3.4.4. These changes were necessitated by changes to the source-line termination conventions used by the INTERCOM EDITOR made by CDC. There would not have been any need for these changes if I were doing all my computing in the batch mode via the central site. Apart from being lazy, I find that working through a terminal increases my throughput, so that it made sense to me to depart from the defacto standard Pascal as distributed by ETH. I know of several universities who made similar changes for much the same reasons. A more serious point is the necessity to empty buffers after each meassage. I would have liked to change the language specification concerning files, but resisted this temptation and changed the operating-system interface instead. Nevertheless, if I want to send an interactive program of mine to someone else, I also have to send the seventy odd changes to the interface and hope that this is enough information to allow the recipient to change my program so that it can run under his system. I will go so far as to say that McMaster offers "standard Pascal", but not the standard Pascal system. I doubt if any "language standard" committee would find it appropriate to consider points such as these which do not affect the language as such.

I have made a point of killing every version of Pascal other than the current one as soon as it has been recieved and tested. Admittedly, the number of Pascal users at McMaster is small, but FORTRAN is very deeply entrenched even amoung the Computer Science Faculty. Within the next few months we are acquiring a second CDC6400 to be run under NOS, so that changes to the OS interface will be required.

To summarize: let us be perfectly clear about what we mean by "standardization". I would like to point out that the University of Toronto is trying to enforce standardization of their SP/k languages by distributing binary modules only and prohibiting recipients from seeing compilers in source form. Do we want this approach? In other words: "Weak or strong standardization?"

Yours sincerely,

N. Solntseff.

NS:ib

P.S. My offer of help still stands. Of the three tasks listed in PNEWS 6, I do not think that I could manage the bibliography properly.

P.P.S. I would like to see a "bug" corner giving details of bugs real or imaginary. One can then know immediately whether the bug one discovers has been noted by someone previously.

OPEN FORUM FOR MEMBERS

PATTERN ANALYSIS & RECOGNITION CORP.
ON THE MALL
ROME, N. Y. 13440
TEL. 315-336-8400
315-724-4072

Mr. Andy Mickel
Editor, Pascal Newsletter
227 Experimental Engr. Bldg.
University of Minnesota
Minneapolis, MN 55455

12 January 1977

Dear Andy:

I would just like to make a short comment on Richard Cichelli's proposal (Newsletter #6) for direct access files in Pascal implemented as long array's. That is, I feel the suggestion was excellent in terms of simplicity and elegance, but that the word "long" is an unfortunate choice. The compiler doesn't have to be told that the array is long or short -- it knows the exact length of every variable. What the programmer is really trying to tell the compiler is that it is all right for the array to be allocated on a slow, mass storage device because faster access speed would not justify (for this particular variable) using the required amount of scarce main storage. Thus, I submit that "slow array" would be more appropriate, as it specifies an attribute of the storage allocation, just as does the word "packed".

Thank you.

Sincerely,

PATTERN ANALYSIS AND
RECOGNITION CORPORATION

By
Michael N. Condict
Programmer

MNC/pak

OPEN FORUM FOR MEMBERS

UNITED COMPUTING SYSTEMS, INC.
A UNITED TELECOMMUNICATIONS CO.

UNITED COMPUTING SYSTEMS, INC., 2525 WASHINGTON, KANSAS CITY, MO. 64108 / 816 221-9700

January 4, 1977

Dr. G. Michael Schneider
University of Minnesota
114 Lind Hall
Minneapolis, Minnesota 55455

Dear Dr. Schneider:

I was impressed by your concern for the future of PASCAL, as expressed in PUGN #6. I also agree with your proposal of the initiation of "proper administration" of PASCAL.

Perhaps, however, it would be wise to include some direction in the areas of P. Brinch Hansen's concurrent PASCAL and Niklaus Wirth's MODULA (if/when it goes). It would be unfortunate if only the"application" areas are "well tempered" and the related "systems" areas are left undirected. Since there is a possibility that we at UCS may be doing some work in more machine dependent areas (using MODULA), it would be advantageous to have a "sounding board" for linguistic adaptations (not necessarily including all machine dependent extensions). Another advantage to this approach would be to help dispense information on how various machine dependent language extensions were done.

Hopefully, this request, if incorporated, would not significantly increase the burden upon the committee. It could significantly increase the scope of PASCAL and PASCAL-like usage, and at the same time, hopefully prove that an "adapted PASCAL' is a good (great) "system level" language.

Your consideration of this matter is appreciated.

Sincerely,

UNITED COMPUTING SYSTEMS, INC.

Larry

L. D. Landis
Distributed Systems Division

LDL/mgr
cc: Andy Mickel
John Strait

(* In a phone call April 11, Larry wished to clarify that he didn't view Pascal as a SIL and rather that emphasis should be placed on MODULA. He urged that anyone releasing Pascal-ware should put it in a source library editor compatible form against which future modifications can be made (such as CDC MODIFY).*)

THE UNIVERSITY OF BRITISH COLUMBIA
2075 WESBROOK MALL
VANCOUVER, B.C., CANADA
V6T 1W5

DEPARTMENT OF COMPUTER SCIENCE

14 February 1977

Andy Mickel
University Computer Centre
227 Exp Engr
University of Minnesota
Minneapolis, Mn 55455

Dear Andy,

I would like to add one more opionion to the standards issue, along with the idea that Pascal might someday replace Fortran.

First of all, there should be no doubt that Standard Pascal will never replace Fortran. Describe Pascal to a numerical analyst and he will laugh. Several mandatory extensions include:

- Parameter arrays of unknown size
- Shared variables for separately - compiled procedures
- Input formatting and improved output formats.

So suppose that we standardize a language resembling Standard Pascal. We lose the time and energy of those making the standard and those modifying Pascal implementations. More people are using the language, so later extension efforts must live by the frist standard. Meanwhile, Fortran programmers look at the language and reject it, so they will never bother with the revised version.

Why shouldn't Pascal be revised? Fortran's main problem is its age. Pascal has been around for a number of years too, so it could benefit from some re-design. Some of the improvements found in Concurrent Pascal, Modula, and Euclid might well be added to the language. (It is not at all clear that the result would still be called Pascal.)

Hopefully, this new language would resemble Standard Pascal to the point that existing programs could be mechanically translated to the new language.

If the hopes of Pascal users are to be realized, it seems that we should recognize the need for a language re-design and work towards the organization of that effort.

Sincerely,

Robert A. Fraley

Robert A. Fraley
Assistant Professor

ABT ASSOCIATES INC.
55 WHEELER STREET, CAMBRIDGE, MASSACHUSETTS 02138
TELEPHONE • AREA 617-492-7100

24 January 1977

Andy Mickel
227 Exp. Engr.
University of Minnesota
Minneapolis, Minn 55455

Dear Andy:

Enclosed is the check I promised for an additional copy of Number 6, and a two-year extension of my subscription. I am looking forward to a long series of interesting issues.

There are several issues I would like to raise which I shall attempt to group together as (1) Standards, (2) Mods to the Standard, (3) Mods to the implementation, and (4) available software.

Now that there is an obvious push for the formation of a Standards Committee in the abstract, I would like to put in a pitch for our interests, and suggest that PUG should be responsible for the organization of the Committee. It seems appropriate to suggest that PUG might reform under SIGPLAN as a Special Technical Committee while putting a first draft ANSI Standard for the language. Any help you might need in getting this ball rolling which I could provide is available for the asking.

Now that I have looked over three suggested implementations of variable dimensioned arrays, it is clear that some mechanism must be provided to review and coordinate comment upon proposed extensions and modifications to Standard PASCAL. My favorite extension is the inclusion of OTHERWISE as the final branch under the CASE statement; a suggestion which I know causes you no end of grief. I have included a revised syntax graph for the case statement which demonstrates the feature.

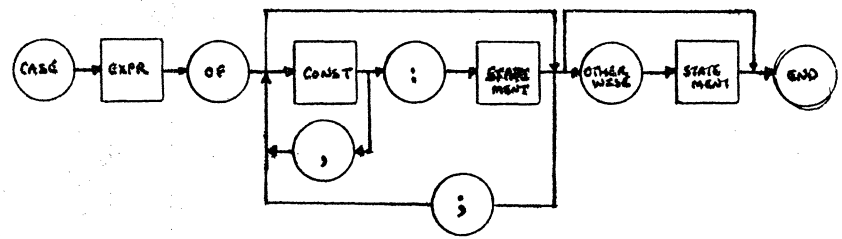
As we have discussed on the phone, the CDC 6000 implementation could be improved. In addition to the proposals made before, I would like to recommend that PASCAL and the system routines be modified so that when files are passed as formal parameters, the FET address is passed, in the same manner as all of the other SCOPE and KRONOS software. Given that PASCAL has an already demonstrated to be inadequate file system, this minor change would allow the user to develop and test new I/O routines without all of the additional calculation involved in adjusting the PASCAL EFET to the 6000 FET addresses. If you have a set of mods which do this, I would greatly appreciate receiving a copy.

The last item concerns available software: I would like to know how we start a register of PASCAL and PASCAL callable software for exchange. For example, I have just completed a PASCAL core dump interpreter which lists specified locations of the user's CM and control point area in octal, COMPASS, alfa, real and integer with a wide range of options. This is useful, obviously, for systems work on a 6000, although the code is interesting. It comes with its own PP routine, BCD, which creates a Binary Core Dump on a user-provided file for later analysis. Given that PASCPMD does not help when the routine being developed is in another language (e.g. COMPASS), being called from a PASCAL mainline driver, this is a real boon.

Looking forward to your action, I remain

Sincerely yours,

Michael Patrick Hagerty
Director of Systems Research
and Design



SPECIAL TOPIC: STANDARDS

The following set of five exchanges regard the topic of Pascal standards. It was first prompted by a very long letter by Niklaus who has come around to the position of conventionalizing some extensions beyond a standard. Niklaus invited Richard Kieburtz and Jørgen Steensgaard-Madsen to reply.

At the Southampton Pascal Symposium I was a late addition to the program for the purpose of introducing a discussion on standards and extensions. The 4 pages are reproduced from the proceedings in order to explain the assumptions made and to report on the reaction. Before my presentation, Bjarne Dacker urged that a consensus be arrived at, rather than a soon-to-be-forgotten discussion. As the discussion began, Tony Addyman (who had informed others of his intentions to get an official (ISO, ANSI, etc.) standard in Here and There PUGN#6) pointed out that perhaps the most important argument in favor of an officially accepted standard is that if trusted Pascalers don't do it someone else will (like a large computer manufacturer) and do it their way!

Another thought that arose during the discussion was that although Niklaus has shown an unwillingness to move in and clear the air, no one would stop him if he did. There was general agreement to this as was the general distaste of creating a "standards committee." So most important (as well as being good news) was that Tony felt that if anomalies in the Revised Report could be fixed up, then it would be relatively easy to work within the British Standards Institute (BSI) to achieve an eventual ISO standard - without resorting to a standards committee. Tony agreed to send in a list of such complaints against the Revised Report. No one at the Symposium objected to Tony's proposed actions.

David Barron who had agreed to Bjarne's suggestion, conducted votes(!) on each of the three items listed under Considerations of a Standard. The first passed unanimously, the second passed with two half-hearted no votes changing to abstentions, and the third passed with only 4 no votes. However in each case approximately one-third of the people present did not vote. I pointed out that I would be seeing Niklaus within a week and would put these ideas to him. The discussion ended leaving people wondering about the future.

When Niklaus came to Minnesota to talk on Modula March 31, John Strait, Jim Miner, Dan LaLiberte and I put these ideas to him regarding standards. I pointed out the need for an officially accepted standard noting the consensus in the Symposium - which surprisingly had not included adding features in the process.

Niklaus and I agreed that I would collect from Pascal users and Newsletter readers suggested topics for necessary clarification in the Report and would work with him on such points so that they could then be included in the Standard. We will also work on a conventionalized set of extensions to be published in a future issue of Pascal Newsletter. It would be nice that if by the end of 1977 these matters were cleared up and that we had an ISO standard. I am of the opinion that real progress without the potential pollution of the language is being made.

So it is really sad to see some people (for example, George Richmond and some of the aspects of his article in this issue) call for more and more redundant additions to Pascal. Sparceness is Pascal's nature (and is a virtue). Anyone who is using Pascal should try to make do with what facilities are already in the language. For example there is so much of a cry to see an otherwise clause added to a case statement. The facilities are already there for a large majority of instances:

```
if selector in [set of case labels]
   then
      case selector of
         .
         .
      end
   else
```

After a period of using and getting to know Pascal, one can conceive of many natural extensions and wonder why these were left out of the language. Answer: a line had to be drawn on the total number of features in order to adhere to another design goal: efficiency of realization!

- Andy Mickel

Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

XEROX

January 31, 1977

Mr. Andy Mickel
Editor, Pascal Newsletter
Computer Center
University of Minnesota
Minneapolis, Mn. 55455

Dear Andy,

I have received the *Pascal Newsletter* No. 6 and would like to congratulate you on a very nice job. By now it is quite evident that the Users' Group and the *Newsletter* cater to a genuine need. Thank you also for your letters inviting me to express my opinion on several issues raised in the *Newsletter* in general and on standardization in particular. The latter is a recurrent topic, although the reasons why a "standard" is needed in addition to the Pascal Report are not entirely obvious.

Standards are successful if, and only if, *many* people feel that each of them can profit by adhering to a mutual consensus, and that deviating from that consensus is detrimental to their individual interests. In the case of Pascal, there is the original language definition, and any implementor must decide for himself whether or not to adhere to it. All too often, he is tempted by his own bright ideas on how to do better on little points, and unfortunately it is the user of his implementation who will later be inconvenienced by the non-standard. But, alas, even the existence of a standard cannot prevent this from happening. I agree that there are a few areas where the temptation to extend the language is particularly strong, and where it might indeed be beneficial to have a commonly accepted way of extending Pascal.

This sounds like a good idea; yet I have my reservation about declaring these extensions to be Standard Pascal. After all, there are many implementations in existence, and it would seem unfair to suddenly declare that what once was a Pascal compiler now suddenly isn't so any longer. Also, once you start on the alley of extensions, there is seldom a consensus about where to stop. An even more serious problem is the published literature on Pascal, which, I believe, would have to be properly updated: the Report, the User Manual, tutorials, books, etc. There is a great deal of virtue in stability.

After these caveats, let me list and discuss those points where I nevertheless believe that a *recommended set of extensions* could have a beneficial influence.

1. Dynamic arrays. It is generally agreed that dynamic arrays are missing and should be made available, even if they cause some conceptual inconsistencies with Pascal's notion of strictly static typing. There remains the question of whether this extension

should bring truly dynamic arrays (as in Algol 60), or be restricted to parameters in procedures (which do not involve any actual storage allocation) such as in Fortran. Should the dynamic property be applicable to named arrays (like in Algol 60), or to variables referenced via pointers only (see SIGPLAN Notices 12, 1, 82-86)? In any case and for good reasons, array bounds must always be static, if the array is a component of a record or a file structure. The proposal of Jacobi (PN 5 p. 23) meets all these considerations, and has proven to be economically implementable.

2. **Array- and Record-Constructors.** The primary motivation is the desire to have a convenient facility for initialization of tables. Yet if a notation for structured values is introduced, it might as well be available in general instead of being restricted to specific places. For example, given a declaration

```
a : array [1 .. 3] of
      record  i: integer; x: real;
                s : packed array [0 .. 4] of char
      end
```

an assignment might look like

```
a := ((5, 0.3, "BEGIN"), (3, 1.2, "END  "), (4, 0.1, "GOTO "))
```

It is of course tempting to admit general expressions as constructor elements, but this may give rise to some nasty pitfalls. Consider, for instance, the assignment

```
a := ((a[2].i, a[3].x, "ARRAY"), (a[1].i, a[3].x, a[k].s), ...
```

From the point of view of implementation, and perhaps also of clarity of exposition, it is reasonable to require a type identifier preceding each list of component values. Yet this appears to be quite cumbersome in the case of structured components of (long) arrays, as in the example above, where the identifier would have to be repeated many times. Moreover, it appears that the use of constructors with type identifiers would necessitate the possibility to include constant declarations after type definitions. This would unfortunately entail a change of syntax. Also, a notation for eliding and perhaps repeating components would appear as desirable. This may suffice to show that the subject of extending Pascal with constructors is a complicated subject. ∎

Let me add two more items to the list which are mentioned again and again in the *Newsletter*. I am, however, rather doubtful about their indispensibility:

3. **Default in case lists.** Certainly there are situations where it might be convenient to have a default case being selected when the case expression is unequal to all case labels, such as when the case expression is of type *char*, and you do not wish to explicitly list all "other" characters. (But *convenient* is not the same as *necessary*).

4. **Formatted input.** The *only* justification for such a facility is *convenience* in reading *densely packed* (i.e. encoded) data. A syntax identical to that of the write statement would seem most natural. But I cannot accept an elaborate proposal like Hagerty's (PN 6, p.43), which includes Boolean values (who would "pack" a Boolean value into 5 characters!) and specifies complicated rules about "overriding decimal". Such sophistications have only one effect: to greatly increase the possibilities of data mismatch errors discoverable at run-time only. It seems that there must be better approaches to this subject than to adhere blindly to the conventions of the past. ∎

Many other extensions have been mentioned as needed, convenient, conventional, or merely desirable. Most of them, however, belong to a different category which, I believe, has nothing to do with the goal of attaining a common language. Rather, their primary objective is to introduce some favourite facility suggested by either a particular application or, more frequently, an existing operating system. Whereas, I have no objection to such extensions in principle, they do not belong into the core language, whose facilities must be understood without reference to any particular implementation. If at all possible, they should be incorporated in the form of predefined procedures, functions, types and variables, and in the documentation they must be clearly marked as facilities pertaining to a given system. There are so many different kinds of operating system facilities in existence, that an attempt to enforce any particular set as a standard would be quite detrimental to implementations with incompatible environment. The version of S. Knudsen (PN 6, p.33) on indexed files is an example: although it may be useful on the CDC machines, it would be ridiculous to enforce this concept on an implementation for IBM computers.

The four items listed above are free of such environment dependent considerations. They migh therefore well be considered -- if properly worked out -- as *Standard Extensions* of Pascal. We might publish a final proposal in the *Newsletter*, thereby avoiding to have to officially change the definition of Pascal as widely published in the literature. A set of Standard Extensions might encourage implementors to adopt a common notation.

The remainder of this letter consists of some miscellaneous comments on various contributions in *Newsletter 6*. Above all, I enjoyed the Southampton-Hobart dialogue and in particular Professor Sales's yuks and ouches (p.61). I emphatically support his advice against private character sets. Two are already too many, but we shall have to live with ASCII and EBCDIC due to higher forces. It is unfortunate that CDC users are compelled to have an additional one based on 6 bits and strange conventions about line ends, and this has considerably hampered transportation of Pascal.

Files and input/output are a frequent topic, and this is not surprising. I agree that files play a special role among Pascal's data structures, and that it would be unwise to try to eradicate or hide this special role by, for example, letting the assignment operator denote the copying of an entire file (p. 61). On the other hand, I disagree with the strong statement that "Pascal's files are an anachronism" (p.47). The Pascal Report specifies clearly that *file* here means *sequential file*, and perhaps *sequence* would have been a less misleading term. The concept of a sequence is as little an anachronism as is the notion of an integer (which retains its importance inspite of the existence of real and complex numbers). In fact, the sequence has so far proven to be the only data structure that is widely accepted and simple enough to describe much about input and output in a machine-independent fashion. Every other attempt has remained highly tailored to specific file systems and proven to be of little interest to programmers not using the same environment. The proposal by Hagerty (p.43) is a nice example to support this case. What, for example, can it mean to "read an

end-of-file", if not some implementation dependent mark in a *sequence* of elements, a mark that may be followed by other elements and is therefore *not* the end. It is important that we distinguish between the end (of a sequence) and a possible way to represent this end. Let not such confusion penetrate the framework of Pascal! An attempt to define any proposal on new facilities in terms of an abstract, consistent set of axioms is a highly recommended test for its soundness and independence of implementation particulars. (See Hoare and Wirth: An axiomatic definition of the programming language Pascal. *Acta Informatica 2*, p. 335-355, 1973).

"Fortran's archaic control character at the start of a printed line" (bottom, p.47) has never been a part of Pascal's definition. It was merely part of a suggested standard for program interchange, alluding to the fact that this convention is used wherever Fortran is available. The Pascal system itself is not even aware of the special significance of the first character.

Another misunderstanding accuses Pascal of being "unsuitable as an interactive language". What, above all, is an *interactive language*? A correct statement might be: "The Pascal 6000-3.4 file system without modifications is inadequate for use in an interactive mode". In fact, the notion of file may well be used to represent the sequence of characters originating at the input terminal. However, at the heart of the problem lies the fact that interactive use inherently postulates *two* concurrent processes, namely the programmer and the programmed computer. Yet Pascal does not include the notion of concurrency. Nevertheless, the problem can be "solved" in this particular case in several ways. The most popular one is to require a *readln* statement before the first input request, which includes a delay until the next line arrives from the terminal.

Finally, I should like to mention that there exist items where standardization should not be considered at all. The form of compiler directives is one of those. They were intentionally moved into comments, so they could be ignored as such if desired. The idea of portability is stretched too far, if even compiler directives should carry over automatically.

Yours sincerely,

Niklaus Wirth

Prof. N. Wirth

c: U. Ammann, ETH

DIKU DATALOGISK INSTITUT KØBENHAVNS UNIVERSITET
SIGURDSGADE 41, DK-2200 KØBENHAVN, DANMARK, TLF. (01) TA 9466

Mr. A. Mickel
University Computer Center
227 Exp. Engr.
University of Minnesota
Minneapoli, MN 55455
USA

February 9, 1977

JSM/HG

Dear Mr. Mickel.

Professor Wirth has asked my opinion with respect to standardizing some extensions to Pascal. He did that after reading our Pascal 1100 User Manual and in his reply he mentioned that you too would be interested. As they may be of interest to others also I have set up a small paper, which you may include in Pascal Newsletter. Further I enclose for you a copy of our Pascal 1100 User Manual.

Yours sincerely

Jørgen Steensgaard-Madsen

COMMENTS ON PASCAL EXTENSIONS

J. Steensgaard-Madsen
DIKU
Sigurdsgade 41
DK-2200 Copenhagen
Denmark

The programming language Pascal was originally designed primarily for educational purposes. Its popularity is steadily growing and it seems natural to consider the language also for applications. Doing so, a few well-known shortcomings of the language increase in importance. From the experience gained by bootstrapping a Pascal compiler to the UNIVAC 1100 machines, and the extensions of the language built into that compiler, I should like to state my opinion on selected topics. These will be

a. initialization of variables
b. dynamic arrays
c. exhaustive specification of parameters
d. the case statement
e. handling of TEXT variables.

Initialization of variables

Quite often a program depends heavily on tables, the content of which is a rather complex pattern of elementary values. With Pascal such tables are held in variables and the initialization of these is cumbersome and probably costly.

As a means to overcome this problem a constructor concept has appeared. A constructor looks like a function, but generally results in a structured value and accepts constant parameters only. The name of the function is identical to an identifier of the result type.

I find the constructor to be only a partial solution to the initialization problem, but perhaps it may be useful as an expression yielding a structured result, i.e. if the parameters may be expressions. To my taste it requires too much writing to build a table with structured entries using constructors. But more important, it means that any table still must be a variable, although its value may remain unaltered after initialization. In this case the table probably will be represented twice, once in the variable and once for the purpose of initializing the variable.

Now, if you provide for the identification of structured constants you have the option of easy initialization by usual assignment. This is done in our Pascal compiler and experience indicates that structured constants are used more in their own right than for initialization of variables.

We have extended the syntax for constant definition by the rules

```
<constant definition> ::=
   <constant identifier> : <type> = <values>
<values> ::= <value> | (<value> {,<value>})
<value>  ::= <constant> | [<subset> {,<subset>}]
<subset> ::= <constant> | <constant> .. <constant>
```

To ease the use of this facility we allow a mixture of constant and type definition parts in one block.

Dynamic arrays

Under this heading two problems must be considered. First, the requirement in Pascal of complete type agreement between formal and actual parameters means that it is impossible to write one procedure to invert matrices of different sizes. This again makes it a doubtful enterprise to build a library of Pascal procedures. No serious technical problems have to be solved to mend this defect. It almost suffices to define a syntax for the description of array parameters without fixing bounds for indices. The most suitable way is to eventually replace the identifier specifying the type of a parameter with a construct like

```
<packed> array [<type> {,<type>}] of <type>
```

Secondly, the fixed-sized representation of values other than file values is fundamental to Pascal, but is felt restrictive to users familiar with Algol to whom it is natural to let the value of an expression determine the size of a table.

I consider it fruitless to modify the type concept of Pascal in such a way that index bounds may be determined by expressions in general. Alternatively I propose to introduce the concept of an array of variables to be distinct from one variable, the type of which is an array structure. The bounds for arrays of variables may then depend on general expressions. Syntactically you may declare several arrays of variables by the rule

```
<variable declaration> ::=
   <variable identifier> {,<variable identifier>}
   [<bounds> {,<bounds>}] : <type>
<bound> ::= <expression> .. <expression>
```

An implementation along these lines is well under way for our compiler and the additional complexity seems modest. The specification of array parameters is taken as a point of unification of the two array concepts.

Exhaustive specification of parameters

The number and types of parameters in formal calls of procedures and functions cannot at reasonable costs be checked at compile time. It is possible to devise means by which the programmer can specify such parameters. Consistency can then be checked at compile time. The requirement in Pascal that formal procedures and functions may be called with value parameters only, can be completely relaxed.

Although this may seem a minor problem I find that its solution is important. Not only just for aestetic reasons. The use of formal procedures and functions is uncommon, but properly done you may achieve protection of data usually connected with a monitor/class/module concept; and with even greater flexibility in certain cases (e.g. hidden and recursive modules). I do have very favourable experiences from actual use but space is too short for further elaboration here. You may consult our Pascal

1100 User Manual for the syntactic details, but this contains no important examples.

The case statement

Pascal has often been praised for its case statement. The explicit labelling of entries makes programs very readable. Nevertheless the Pascal Report does not specify the action when the case expression does not compute one of the values labelling an entry. It seems most reasonable to provide the programmer a means by which to handle this situation and settle for a common interpretation if that is not used.

Further I find it very convenient but not so important to allow an interval to indicate labelling of an entry. In our compiler we have adopted the following syntax

```
<case statements> ::=
   case <expression> of
      <case list element> {;<case list element>}
   <case termination>
<case list element> ::=
   <case labelling> {,<case labelling>} : <statement>
<case labelling> ::=
   <constant> | <constant> .. <constant>
<case termination> ::=
   end | otherwise <statement>
```

The statement following otherwise is executed if none of the labelled entries are selected. The termination end is equivalent to otherwise <empty>.

Handling of TEXT variables

A format specification in the read procedure in analogy with the write procedure has been claimed a need. I have no strong opinion on the subject itself but want to warn against rushing to a solution. Proper use of the read procedure seems to be difficult, judged by the number of errors found in beginners programs. The reason to this is, I suppose, the lack of a suitable standard structure in TEXT variables. This may be explained by the development history of Pascal, especially the late addition of a read procedure including type conversion.

I would find it a most unhappy situation to introduce a standard for formatted input before an agreement on TEXT structure. This should include a rule stating that the value of eof only changes when a line marker is passed, in complete agreement with the scheme

```
while not eof ( f ) do begin
   while not eoln ( f ) do begin
      read ( f, x ); use ( x )
   end;
   readln ( f )
end
```

Another trouble with reading a TEXT variable is that the above scheme is only correct if x is a CHAR variable. A closer look into the problem reveals that the above scheme applies to reading in general, if trailing blanks in a line are skipped during a read operation. This will be true if x is replaced with several variables provided that read (f,v_1,v_2,...,v_n) is considered a fatal error only if eof (f) is true prior to the call.

With the above structure, format specification may be safely introduced if interpreted in such a way that detection of a line marker may shorten a field.

Formatted reading would probably be used mostly to read a TEXT variable previously written with Pascal output format specification. Writing may result in a field larger than specified. This situation ought to be detectable when reading and a standard structure of TEXT variables may be a sufficient means to this.

Concluding remark

Except for formatted reading my opinions expressed above are based on actual experience, both my own and a large number of computer science students. Neither in themselves nor in combination do the presented extensions complicate the compiler seriously and the additional conceptual complexity is clearly outweighted by the increased possibilities.

(* Jorgen is not a PUG member yet. *)

State University of New York
at Stony Brook
Stony Brook, New York 11794

Department of Computer Science
telephone: (516) 246-7146

StonyBrook

March 7, 1977

Professor Niklaus Wirth
Xerox-Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

Dear Professor Wirth:

I received the copy of your letter on standardization of Pascal extensions, and read it with much interest. I certainly agree with the general philosophy expressed on standardization and on the role of language extensions. The concern that you express for keeping the published literature up to date with the working versions of the language is also well taken; no implementor will ordinarily undertake such a responsibility.

My principal concerns on the Standard version of Pascal, as opposed to possible extensions, are two:

A. The complete typing of formal procedure parameters should have been standardized. The strong typing inherent in the language fails to carry over here, leaving to the implementor the choice as to whether to omit type checking of parameters in calls upon formal procedure or function names (unthinkable) or to compile a run-time checking mechanism. Several implementors have chosen a third route--to define their own syntax for the declaration of the parameter types of formal procedures. These several syntactic devices have in common that they all permit full type checking during compilation, but they are mutually incompatible.

B. The field width specifier given in the arguments of a call to the procedure Write should not have been made a part of the Standard language in my view. Although the mechanism is convenient and easy to use, its syntax is not context-free and it is not easy to implement unless one uses recursive-descent parsing, which allows the use of semantic information to aid the parse.

With regard to the recommended set of extensions, it is really important to look for extensions that work well with the Standard language and work well with one another. My own prejudices are that one should first add diagnostic facilties to an implementation of the standard language, then look at extentions. One obvious extension, consistent with the design, is to relax the implementation restriction on the maximum cardinality of set types. There are known algorithms for flow analysis, for instance in which the ability to handle large sets as bit vectors is crucial to performance. Next in order, by my preference, would be to add typed, structured constant declarations. The lack of a facility to initialize tables is a genuine weakness. Third would be an external, or separate compilation facility, with type checking extended across program linkages. Fourth on my list would be some form of string processing, at least at the level that exists in ALGOLW, but preferably allowing a variable length specifier in the sub-string selector. On the recommended set of extentions that you have addressed:

(1) Dynamic arrays are greatly overrated. Fast, dynamic sequential-access storage can be provided by implementing internal files as corefiles. Linked lists are convenient and easy to use to implement stacks and queues. In any single program, vectors and matrices tend to be all of the same size, or of a few fixed sizes. In fact, until one has a facility to compile program components separately, creating a procedure or module library of relocatable programs, there is really no need at all for dynamic arrays. When a library module is designed to be incorporated into a variety of applications programs, then elastic bounds do become somewhat of a necessity. However, a very minimal degree of 'dynamism' is required, even in library modules. Each applications program will have a characteristic set of dimensions that its arrays use, and these dimensions, communicated to the library module during program linkage, should provide sufficient elasticity in the bounds of arrays. I would propose that the critical dimensions be declared as parametric constants not known at compile time, as distinguished from "manifest" constants. Parametric constants could be used to define subranges, just as are manifest constants. The only limitations that I know of on the use of parametric subrange types would be in connection with another possible extension, default clauses in case lists. Of course, arrays and sets indexed by or based upon an elastic subrange would have to be doped in an implementation, even though the constants are defined prior to actual run initiation.

(2) Array- and Record- constructors. If the initialization of tables is the primary motivation for these constructors, and I believe that it is, then why should they be defined on the right side of an assignment statement, in the statement body of a block? It seems that it would suffice to define typed, structured constants in the block heading, allowing type and const definitions to appear in alternate order. I certainly agree that the inclusion of general expressions as constructor elements would be a major, and I think unwarranted extension of Pascal. The notion of multiple assignment has some attractive aspects, but this is something to be built into the foundations of a programming language, not to be added on.

(3) Default in case lists is only a convenience, a cosmetic extensions, but it turns out that it is a very attractive one. Our students, who are currently using the Pascal 1100 compiler from Copenhagen which has this feature, like it a lot. And we haven't encouraged them to use it, they have just found out about it. Since it is one of the easiest extensions to make, perturbing nothing else (until parametric constants are added, at least) I see no objection to it.

(4) Formatted input has no justification that I can see. It has a historical origin in the technology of fixed-field, unit-record data processing. Far more useful in some large-program applications would be the ability to specify, in a machine-dependent way, the format of data packing in the declaration of a packed record. This would be a rather specialized extension, but is worthy of some serious thought.

In our implementation, we have first concentrated on providing some useful diagnostics, borrowed with much admiration from Ed Satterthwaite's contribution to the ALGOLW compiler, and on a relatively clean implementation that can be built upon and maintained. We have given a lot of thought to generating efficient code, and have provided hooks for optimization, but have deferred work on an optimizer until we feel that most of the compiler bugs have appeared and been eradicated. We are just now completing the implementation of nonstandard files. An efficient and safe storage management scheme is our next target, and doesn't appear to be too difficult. We are presently working on two actual extensions, separate compilation with type-checking across module linkages, and the addition of typed, structured constants. Incidentally, we adopted your suggestions for the syntax of record and array constructors. Some of this may be working by the time of your visit.

We have thought about adding a default case clause, and allowing ranges to specify case labels, but I'm not sure this will get done. Also, we have thought about allowing parametric constants, but no decision has yet been made on whether to go ahead with an implementation of this idea, either.

One final comment on the use of Pascal in writing programs for interactive execution is that the expedient adopted in the Copenhagen Pascal/1100 seems worthwhile. They have made the nonstandard implementation adaptation that when a Text file is opened, or Reset, the Eoln condition is initially true. This means that to accept input, one may need to execute an initial call to Readln, but it avoids the condition that input from a terminal is expected prior to printing of the first prompting line of output. Thus it works very nicely in interactive programs.

I look forward to an in-depth discussion with you at the time of your visit on March 23, until then.

Sincerely,

Richard B. Kieburtz

Richard B. Kieburtz
Professor

RBK:pdm

Third Annual Computer Studies Symposium "PASCAL - the LANGUAGE and its IMPLEMENTATION" University of Southampton, March 1977

THE FUTURE OF PASCAL (Extensions and Standardization)

Andrew B. Mickel

University Computer Center
University of Minnesota

The Present State of Affairs

1. It's been 7 years since Pascal's initial development, but only 3 years since Pascal has seen widespread use and easy to obtain literature (books) has been available.
2. We have 3 official documents:
   - the Revised Report (Second Edition, Third Printing of the book: Pascal User Manual and Report)
   - the Axiomatic Definition (in Acta Informatica, 1973)
   - the User Manual
3. Now in early 1977 we have working implementations on dozens of different machines, thousands of users, an ever-increasing base of computer science departments which are using Pascal for teaching, and a rapidly growing user's group of more than 800 members in 28 countries.
4. Pascal has had an enormous effect on computer science - just witness the imitations of features in the literature and in conference papers.
5. Much applications and production software is being written in Pascal at all levels: from individuals, to small software writing firms, to large organizations (research computer centers and corporations.)
6. No major computer manufacturer has yet officially produced and supported a Pascal system for general user applications.
7. We therefore can proclaim a fair measure of success. But...
8. Even though we have a "standard" in the official documents, many implementors are not adhering closely to it. There are at least these reasons:
   - most implementations so far are done at universities and it is their purpose to experiment with new things: some valid, some "bright ideas."
   - most Pascal compilers are written in Pascal and are very transparent and compact which allows easy modification.
   - there are lots of questions about details not specified in the 3 official documents but which are left up to the implementation to decide what to do. This aspect has led to accusations against Pascal such as CDC bias, requiring a look at the CDC compiler to see how it did things (things defined by implementation), and whom to ask to find out about other aspects.
9. The result is bad not only for users of these Pascal implementations, but also for the implementor and the future of Pascal if it is to spread in the world's computing

community. Increased acceptance of Pascal helps each Pascal programmer to be able to use Pascal respectably.

10. Within the Pascal community (PUG in particular) we have some of the best people in computing in the world today. They naturally tend to strive for excellence and would like to see the looseness tightened; the vagueness clarified.
11. Thus the issue of an officially accepted standard for Pascal is raised which supposedly should help the situation.

Desirable Goals and Current Problems

1. Let's consider the original design goals of Pascal:
   - sparse, simple language (easy to learn/efficient to translate)
   - general purpose language (but not all-purpose)
   - vehicle for portable software (certainly better than FORTRAN)
   - tool for systematic programming (teaching and writing reliable software)
   - efficient realization (for sheer practicality)
2. What roles should Pascal play?
   - an as-low-as-you-want-to-go high-level language forming a basis for much other software
   - help put an end to the FORTRAN age so that young new programmers won't be faced with a life sentence of writing ugly code because of "practical realities." Using Pascal should be a respectable activity.
   - to be an alternative to dinosaur languages. Consider how hard it is to get a common medium such as Pascal widely established. There's not much hope for another language to come along if Pascal (even with its small imperfections) doesn't make it. Computer manufacturers will continue to control users' lives by pushing FORTRAN, COBOL, and PL/I with the excuse "that's what the customers want."
3. It is wise to stay within these assumptions (upon which much of Pascal's current viability lies), and try to understand how to satisfy all the design goals to the greatest degree without holding any single one to an extreme degree.
4. Within Pascal User's Group, many persons are quite concerned that Pascal will fly apart, be killed, or become just as bad as other languages without adherence to a standard.
5. But there is also pressure to use the movement for a standard to extend Pascal at the same time. This is bad because:
   - most importantly, there is a current investment in documentation in the form of defining documents, manuals, and books; implementations which are currently operational; and applications software. We are already in cement, and it is too late to add extensions to the official language.
   - the size of the language as described in the Revised Report is already large enough in terms of learning the whole language in a reasonable time and in writing complete implementations on small (mini/micro) computers. The importance of small computers cannot be overlooked because of their ever increasing numbers and use by more and more people.
   - a committee, which would have to effect a standard,cannot possibly possess the clarity of vision of a single designer who alone considers design goals and tradeoffs.
6. There is much difficulty in obtaining an officially accepted standard by a standards organization. For example, I am told that ANSI requires a committee. Who would choose it, how will it meet, what will its powers be, and how binding will be its decisions?
7. The basic problems with the three official documents seem to be semantic holes (Swiss cheese?). On the other hand their outstanding virtue is small size. The Revised Report requires reading between the lines, the Axiomatic Definition goes as far as it can but is not complete, and the User Manual is not a rigorous source for semantics and shouldn't be. The problems are vagueness and uncertainty. The situation might be better today if i) it had been explicitly spelled out what features were left out from Pascal and why, and if ii) it had been explicitly stated which unspecified details in the Report were left up to the implementation to define and suggest valid alternatives. We would then know where we stand as users and implementors, and be saved from the archeological digging of trying to find these things out.

Considerations of a Standard

1. The case is now made for:
   - standardizing the Revised Report with semantics tightened up,
   - conventionalizing extensions to the standard which apply to any implementations incorporating them, (The User's Group and Pascal Newsletter can be the forum.)
   - stating examples of extensions which should not be conventionized.
2. The advantages of an officially accepted (ISO, ANSI, etc.)standard are:
   - If most people involved with Pascal adhere to it, it will become a living standard and there will be peer pressure (political enforcement) brought to bear against others. Therefore users can point to implementations masquerading under the name Pascal and avoid them. At the present time it would seem that the Revised Report would be such a standard, except for the fact that so many influential Pascalers find it hard to defend mostly because of semantic holes.
   - Portable software possibilities are enhanced, and users are happy.
   - It will increase acceptance of Pascal by large organizations because Pascal will appear to be a legitimate option to take for writing software.
   - It will be economically enforceable in the marketplace. If a large customer (say the United States Government) wants an ANSI-standard Pascal in the manufacturer's array of software, it will be there.
3. Some extensions should be conventionalized and others should not. Many

implementations do provide desirable extensions, some of which enhance Pascal's utility. It is possible to make these uniform across computer systems because they meet many of Pascal's design goals. The spirit of conventionalized extensions is: "if an implementation extends in a certain direction, it should do it this way." On the other hand, many extensions should not be conventionalized because they are so machine and operating system dependent as to conflict severely with design goals (mainly efficiency).

4. An incomplete list of details that seem to need attention regarding the Revised Report:
   - the symbol ".." is a Pascal symbol in the User Manual, but not in the Report
   - sets of char are not necessarily guaranteed, but in practice seem to be a useful guide for minimum set size.
   - what is meant by the concept of same type?
     a) explicitly same type identifier? or
     b) same structure?
   - are compound boolean expressions evaluated fully or sequentially left to right allowing partial evaluation? (specified in the Manual but not in the Report)
   - what should be the undefined values of scalars and pointers? (nil for pointers will not necessarily suffice.)
   - what is the effect of a case value out of range?
   - what is the effect of unset tag fields in variant records?
5. Candidates for conventionalized extensions:
   - variable extent array parameters
   - constructors for structured constants
   - specification of all parameters and their types for procedures and functions as formal parameters.
   - a data initialization facility (value part)
   - formatted read procedure
   - reading and writing enumerated scalar values
   - the procedure dispose
   - external procedures and functions (whether precompiled or in source form)
   - interactive input/output
   - otherwise for case statements
6. Examples of extensions which should not be conventionalized across implementations:
   - operating system file substructures, their access methods, and their myriad attributes (including direct access secondary storage)
   - additional, predefined constants, types, variables, functions, and procedures
   - compiler options
   - very specialized extensions (significant digit arithmetic, error trap labels, extra looping control structures, synonyms for standard Pascal symbols, etc.)

Manchester University
Oxford Road
Manchester
7th April 1977 (*late at night *)

Dear Andy

Please find enclosed an attention list, which refers to the Revised Report, for you to put in newsletter #8 and ultimately to pass on to Niklaus Wirth. Many of the points may seem trivial, but I am trying to prevent problems later.

I have had no time prior to Easter to take any action on standardising Pascal in the UK, apart from generating this list. This list includes contributions from others. I will be sending you a copy of a letter which Brian Wichmann has sent me on this matter. Some of the items on this list are due to Brian.

This letter will have to be brief, since I am trying to type it myself on an online terminal (offline)! As well as the list!!

I will be putting a case at the next meeting of DPS/13, (the British Standards committee dealing with programming languages) for the production of an official standard for Pascal. This may be either a UK standard or an ISO standard. If this case is accepted then I can form a working group to study the problem and DPS/13 will apply to a superior committee for facilities for the work. It is the responsibility of this committee (DPS/-/1) to initiate new work. The working group will be OK, since it wont need BSI resources.

The working group (roughly equivalent to X3J1-4's in ANSI) will have two main tasks,

1. To critically examine the current Revised Report and submit their results to Niklaus Wirth (via you).
2. To ensure that any document(s) resulting from your work will be acceptable as standards documents. I dont want TWO standards for Pascal.

The WG will have the responsibility for producing a draft standard as far as the standards organisations are concerned.

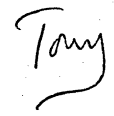
Some thoughts on standards.

1. In addition to the standards document for standard Pascal, we need :-
   a) A definition of acceptable alternative representations of the Pascal special symbols e.g. ↑ []
   b) A suite of programs to validate Pascal compilers for conformance with the standard.
2. Here is a possible definition of a standard conforming processor. (processor = compiler + run time support or interpreter etc.)

A standard conforming processor must correctly process all standard conforming Pascal programs. In addition, it must be able to determine whether or not it's input is a standard conforming program. This has implications for extensions. A processor must be able to monitor the use of any non-standard facilities or semantics. This monitoring could be optional. This also implies that the standard must not forbid anything that cannot be(easily) checked for.

If anyone in the UK is willing to assist in the production of the attention list and/or join in the WG, would they please contact me?

Yours, Tony Addyman.



AN ATTENTION LIST (PART 1 ?)

Chapter

3 .. as a special symbol?
forward as a special symbol ? (see 10)

4 Strings are constants of PACKED arrays of char - but see also 8.1.4 and 12.3.8

6.1.2 a) Ordinal values of type char - digits are coherent?
letters & digits are ordered?
b) Using ISO or similarly full character sets,are the control characters e.g. FF,CR etc. members of the type char?
If yes - do they have a constant representation?
- is LF (the end of line marker) to be a member of the type char? - see 6.2.4 and 12
If no - are they to be all converted to spaces on input?
Including the one chosen to be EOF marker?

6.1.3 NOT REAL ! Introduce the concept of the Associated Scalar Type from the User Manual.

6.2 Packed has no effect on the meaning - but see 8.1.4,9.1.1, 9.1.2 and 12.3.8

6.2.1 Restrict index type to scalar(except REAL)

6.2.2 Add "A record type containing variants may hold the value of only one variant at any time".This is a (poor) attempt to say something about the storage of variants,and whether they overlay.

6.2.3 SET OF REAL ? See also the comments on set operators and type equivalence.

6.2.4 Are file components allowed to contain pointer values? If so, is this sensible?

6.3 TYPE PT = ↑TEXT ; ???
Pointer equality tests are allowed - but see 8.1.4

7.2.1 The meaning of a program with an index expression out-of-range?This should be ILLEGAL not UNDEFINED.
Note - some people think packed arrays are not indexable.

7.3 What is the meaning of P↑ ,if P is undefined or NIL? Could this be an ILLEGAL program,please?

8 The meaning of [X..Y] if X > Y ? - see UM-8
Can we deduce the base type of a <set> from the types of its elements? If so,then [1,2] is of type SET OF INTEGER.
The values 1 and 2 are constants of type integer - see 4
Add comments on integer arithmetic and MAXINT from UM-2C
Add comments on boolean expression evaluation from UM-4A
Introduce the concept of Associated Scalar Type from UM-5B

8and8.1 Operations between sets and the use of IN ; consider :-

```
TYPE S1 = SET OF 'A'..'Z'; S2 = SET OF '0'..'9';
VAR LETTERS : S1 ; DIGITS : S2 ; CH : CHAR;
.
.
IF CH IN LETTERS+DIGITS THEN .....
```

Is this legal?No ! - the types are incompatible.The definition of sets and set operators must be phrased to make this legal.

8.1 MOD and DIV on integer subranges?

8.1.4 Equality on pointers? - see 6.3
Operations on (packed) arrays of char - see 4 etc.

Chapter

9.1.1 When are two types identical? If same type identifier then see the set example etc. If same "structure" then consider these :-
Assuming

```
TYPE A1 = ARRAY[1..10] OF INTEGER;
     A2 = PACKED ARRAY[1..10] OF 1..100;
     A3 = ARRAY[1..10] OF 1..100;
     R1 = RECORD A : CHAR; P : ↑R1 END;
     R2 = RECORD B : CHAR; Q : ↑R2 END;
     R3 = RECORD F : TEXT END;
VAR A : A1; B : A2; C : A3; D : R1; E : R2;
    F,G : R3; H : 5..10; J : 3..8;
```

Consider A := B; B := C; D := E; F := G; H := J;
Note - File assignment is not prohibited (yet)
- Assignment between variables of different subranges of the same type is not explicity allowed.
A program which assigns an out-of range value to a scalar or subrange variable should be ILLEGAL.
What about the assignment restrictions from UM-10(page 64)? These need run-time checks.Is a compile time restriction possible?
The order of evaluation of <variable> and the <expression> should be UNDEFINED.Why should side-effects from a function have a defined effect?

9.1.1.1 Do expressions of type subrange of integer exist?

9.1.2 The assignment rules must apply to value parameters.How about VAR parameters? File parameters by value prohibited?

9.1.3 GOTO into a structured statement - OK?It must be prohibited. If merely undefined ,a program which compiles successfully and is run with all checking on can still go "wild"!

9.2.2 Case label types? scalar(not REAL)?
<case label> ::= <constant> ?
Action on case expression out of range?Could this be either a) the empty statement of ALGOL 60 or b) ILLEGAL?

9.2.3.3 A non-local variable as a control variable?
The semantics in 9.2.3.3 do not cover
FOR I := 2 TO 1 DO S;
The value of the control variable should be undefined on exit, see UM-4C3.This suggests positive action to store an "unpleasant" value in the control variable - GOOD!
Consider UM-4C3,if the final value is calculated once only it is impossible to change it,so why prohibit any attempt to do so?Assignment to the control variable should be illegal - but how to check for it? (Non-local references from procedures). Is the order of evaluation of e1 and e2 undefined?

9.2.4 The effect of nested WITH statements? - see UM-7A.
WITH A,B DO ≡ WITH A DO WITH B DO .
Why prohibit the alteration of i in WITH A[i] DO ?It is very difficult to check.Could the WITH statement be defined to evaluate its <record variable list> just once?It would then be compatible with VAR-parameters and the FOR loop. (see 9.1.2)

Chapter

10 The rules of scope and the accessing of non-local variables and types etc. are not adequately defined.(see UM Introduction and UM-11A).There is NO mention of defining before using!!!
So consider these :-

```
TYPE A = RECORD etc.
PROCEDURE ...
     TYPE PA = ↑A;
          A  = RECORD (* DIFFERENT *) etc.
```

Which type A do variables of type PA point to??

```
PROGRAM ...
     PROCEDURE P(...)
          PROCEDURE Q(...)
          .
            P(...)  (* but which one ? *)
          .
          END
          PROCEDURE P(...)
          .
          .
          END
     .
     END
END.
```

Which procedure does Q call?The rules of scope suggest the second one.

10.1.1 Is assignment to f↑ allowed if one is reading from f?
Is put(f) allowed on file f if eof(f) is true because of reading to the end of the file,without calling rewrite(f)?
Is skipblanks (see UM-12A) legal?

10.1.2 Has dispose definitely been down-graded to a pre-defined but not standard procedure?

11 What is the effect if no assignments occur dynamically inside the function to the function identifier?An undefined value is returned?

11.1.3 Definition of trunc and round should be taken from UM-2B.
Please define ord(user defined scalar) to start from 0.

11.1.4 What is the effect of pred(i), if i = 1 and var i:1..10?
It should produce 0 without error since pred(in this case) produces an integer value.But what about pred and succ on user defined scalars?Should it be a fault?

12 Is reading a subrange variable subject to the same conditions as assignment?what do read(integer) and read(real) do if eof is true before read is called?How about returning an undefined ("unpleasant") value?

12.3.6 Action if n = 0? Compare this section with UM-12B7 and UM-12B3

12.3.8 see 4 and 8.1.4

14.4 This must remain as a program interchange consideration only. Implementations of Pascal should not have to 'know' about this if their operating systems do not believe in it.

UNIVERSITY OF MINNESOTA
TWIN CITIES

University Computer Center
227 Experimental Engineering Building
Minneapolis, Minnesota 55455
(612) 373-4360

April 24, 1977

Tony Addyman
Department of Computer Science
University of Manchester

Dear Tony,

Thank you for going to the trouble of making the list of potential problems in the revised report. I'm impressed at the thoroughness evident in your list. I'm printing the list as an example, and what I will do is collect those sent to me by others, sort and combine them and then send them on to Niklaus as he and I agreed. We want to get most of this done by September.

Please don't forget the principle we learned as of Newsletter #5 (Wirth's letter): don't confuse the language with the implementation. Also remember that because the revised report is concerned with the language only, some aspects of Pascal are intentionally left undefined to be defined by the implementation. But there definitely should be a list of specific aspects to be defined by implementation accompanying the revised report, rather than a vague implication by omission.

Omissions in the revised report right now can mean:
1) the aspect in question is undefined,
2) the aspect in question is to be defined by implementation, or
3) the aspect was not given consideration and the revised report therefore has as error.

OK, so I (for example) don't think that including 6.3 in your list is a valid complaint. The language doesn't prevent pointers to files, files of files, etc. and it shouldn't. An implementation (with today's technology) may have to restrict these possibilities.

I know you are going ahead with standardizing via ISO; it will certainly be a far sight better than even touching ANSI. I've just found out more details on the BASIC Standard and it was very disappointing to nearly everyone I talked to. There were a lot of 8-7 votes (some going the "wrong" way) in a 15 person committee. And the whole effort will be measured in units of years. So, I emphasize again that committees are a disaster, and the one you need for BSI and ISO is only for review as you promised. In other words your working group is not to twiddle with the language. You did say when I was in England that there were precedents within ISO for creating standards without committees. That is the only acceptable route for Pascal at this point. I want to be able to trust you. I hope you will do your best.

Keep smiling,

Andy

The University of Tasmania

Postal Address: Box 252C, G.P.O., Hobart, Tasmania, Australia 7001

Telephone: 23 0561. Cables 'Tasuni' Telex: 58150 UNTAS

IN REPLY PLEASE QUOTE:

FILE NO. .....

IF TELEPHONING OR CALLING

ASK FOR .....

DEPARTMENT OF INFORMATION SCIENCE

28th January, 1977.

Mr. Andy Mickel,
University Computer Center,
227 Experimental Engineering Building,
MINNEAPOLIS, Minnesota 55455 USA.

Dear Andy,

Please find enclosed a contribution for the PASCAL Newsletter in some future issue. It addresses the file question; quite a serious one for PASCAL. If I can reiterate something I wrote in it to emphasize it to you, I believe that PASCAL has more to fear from its friends than its enemies.

I'd also like to briefly comment on your editorial in #6 where you said you couldn't understand my views on page 2. It is very hard to say all that one would like to when writing is all that is possible across several 1000's of kms. What my attitudes are briefly are as follows:

(1) Adhere to standard PASCAL where this is well-defined in the Report or where a portability trend can be clearly perceived. Unfortunately the Revised Report is hopeless as a standards document (much too loose, and dumb on many semantic issues), and PASCAL is inadequate in some areas.

(2) Where there is a gap in PASCAL, or an unsupportably bad feature, then if the gap has to be filled it should be with (a) maximum compatibility with PASCAL aims and style, and (b) maximum compatibility with Burroughs practice. Somewhere a compromise; though often the two agree. The sort of thing I have in mind is in the specification of file attributes (none in PASCAL), or compiler options (too terse and clumsy in Wirth's PASCAL), or extended standard functions (even Wirth has a larger set than the Revised Report).

I could weep over some of the things PASCAL has in fact carried over from the past (its silly semicolon structure for example), but no-one can do anything about them now. I wouldn't bother trying, except to point out the mess, and apply a bit of plaster in our implementation to ease the problem.

I wouldn't think this covering letter is worth including in the newsletter (I don't feel too slighted), but you may if you wish.

Yours sincerely,

Arthur Sale

Arthur Sale
Professor of Information Science.

PASCAL files

In PASCAL Newsletter #6, I made some remarks concerning the inadequacy of the PASCAL file concept. Provocative, perhaps, because I have drawn a number of letters defending PASCAL and suggesting extensions to it. In fact, Newsletters #5 and #6 also had comments by other people pointing out possible extensions to PASCAL in this area. I think the topic is so important, judging from the interest and the many suggested remedies, that it deserves a brief comment in the Newsletter. So here goes.

(1) Are PASCAL's files inadequate?

That depends, of course, on how you interpret inadequate: inadequate for what? I put the question in terms of the use of PASCAL for systems programming, and as a possible user-programmer language (the FORTRAN replacement role). I would have thought the answer was quite clearly no; for instance I could not write an analyser in PASCAL to inspect the code-files of the B6700 computer (that requires random-access), nor to scan a disk directory; it would be unbearably cumbersome to carry out any conversation with an interactive terminal as the discourse would have to be carried out at the read(char) level....

Some of my correspondents disagreed, and thought PASCAL's files were just fine; a sequence of elements was all they needed. If so, fine, it can be enough for teaching and some applications. However, in nearly all cases they gave themselves away subconsciously by proposing far-reaching changes to PASCAL which would go far outside the current language. Often these were disguised as innocuous extensions: let files be treated as full PASCAL types.... No, it is widely recognized that the files of PASCAL, though quite adequate and regular for a teaching environment (the design target of PASCAL) are not fully up to the reality of the computing world.

(2) Are files variables?

I have argued that files are not variables in the same senses as scalars, sets, records and arrays, and that it would have been better for PASCAL had the declaration of file objects been separated from that of VAR objects, just as CONST and LABEL are. I shall have to justify this view later, though it is by now impossible to make such a change in the language.

This view is the one to which most people take umbrage, and they usually state that files are variables, with equal status with other variable types,

following this up with examples of how files may be used in PASCAL as full variable types. To quote from one letter:

" a) "file1 := file2" should specify a file copy.

b) An array of files could be an array of (pointers to?) file descriptors in main storage.

c) "file1 < file2" is just as meaningful as "array1 < array2" or "'cat' < 'dog'" and could be implemented as a small (albeit time-consuming) loop.

d) The scope of files could be the same as the scope of variables (procedure entry and exit). Of course a file declared as a formal parameter to a main program should exist (or be created) before execution and after termination."

I would not attempt to argue that the above could not be done; I could easily see how to do these things myself. The mind of man is quite capable of thinking up a meaning for any construct. No, my quarrel is that these views are very superficial. For the sake of one regularity (treating files as full variables) they would import into PASCAL a whole host of other second- and third-order irregularities. Let me remark that the ideas I have quoted above must have occurred to every serious PASCALler; they must have occurred to Wirth; surely it is significant that the Revised Report is so quiet on this subject? Let me try to show some of the flaws in the reasoning.

(a) Is "file1 := file2" sensible? The first problem is that some files are read-only (a card-reader?) and cannot be assigned to. Also some files are not of finite length (a file equivalent to a remote-terminal for example) and the copying might be infinite. Then there is the problem of the initial and final states of the two files. Suppose file1 and file2 have something on them already. Does the statement imply a reset and rewrite followed by the copy? And how are the files left? positioned at eof, or closed, or reset/rewrite called?

(b) Is "file1 < file2" sensible? Much the same things could be said. It is easy to define the ordering if the files are of different length, but what if they are empty (never written to) or never opened? In what state are they and their file buffers left? Since some things are inherently not ordered (sets, records), only some files could be compared. What to do if the file components were records with variants? This is regularity?

(c) Is an array of files sensible? Sure, one can have an array of files, and a record with a file component. It is easy to see the logical consequences here: they lead from allowing files as structuring components of arrays and records, to allowing files as value parameters, probably even to allowing records, arrays and files to be function result types, and finally to the ultimate absurdity: allowing files of files. Remember that any operation involving files and file assignment must cause a copy of the whole file; it is not sufficient to copy the descriptor. Even more so must it be the case that writing file-descriptors to a file will lead to chaos as time elapses and some of the objects described vanish and others change...

I will say it again: the confusion arises because files are something outside a program in execution: their lifetime (or extent, to use a technical term) is not identical to their scope.

(d) What about scope? If one views the scope of a file-name as the region in which it is known, then there are no problems about associating the scope of a file-name with the scope of the program/procedure/function in which it is declared. This is exactly the interpretation in B6700 PASCAL with the additional semantic interpretation that at procedure exit all files still open in that scope are implicitly closed, with the consequent side-effects.

It is silly to claim the program heading of PASCAL as a solution to this problem. A little familiarity with CDC computers reveals it as a kludge, and an importation from CDC FORTRAN. The "parameters" in the program heading are not PASCAL parameters. Though this would be preferable, and would remove the irregularity of the program heading itself, it would not solve the problem since it does not address it adequately. (Quite as a side-issue, why cannot main-programs be procedures, thereby allowing them to be called as things with genuine parameters? Any answers?)

Come back and look at lifetime. What sorts of files are there? Some have lifetimes which precede the program's life in execution and continue past it. Some permanent files for example on disk, some tape files, a remote terminal file, and so on. Others do not exist before the program starts execution, but exist after it: a disk file written by the program for example. Yet others exist before the program starts

execution but not after it (as in an archiving program's usual handling). Some only exist during the program's lifetime and are quite temporary. And others, for example print-files, are created during the program's execution and are then detached at the point of closure, to live on for a brief time (inaccessible to the program again) until they are printed. Surely lifetime and scope of files are orthogonal concepts? If they are not, then we get all sorts of difficult and really messy problems. Let me detail a few.

(i) Suppose I have a compiler, written in PASCAL, and it needs provision to talk to an interactive terminal user who is using it to compile something. Fine, you say, the remote terminal is an external file imported into the program. Declare it in the program heading. Yes, but this compiler is also used in queue (batch) situations. Though it knows of this file, it never uses it in these situations, so it never opens it, so it never exists for it. Declaration in the program heading might send us on a fruitless search for a non-existent file we were never going to access...

(ii) Suppose again I have a program writing a file. During the course of execution, it knows that the file it is writing is rubbish because errors have occurred. It doesn't want to enter it into the permanent directory. But if its ok, it does. How? In existing PASCAL?

I can keep on going. I hope these are enough examples to get you to supply some more of your own which highlight the difference between scope and extent. A file ought to be an object whose lifetime is controlled (if at all) by explicit program commands, but whose name is known in a given scope.

(3) Is the best way to random access through slow array of...?

A key question, if you accept the importance of being able to randomly access files at all. The answer must be no, however, for exactly the same reason that sequential files are not slow array of (char?). Both entities are not variables in the same sense as the rest of PASCAL, and both entities may be of unknown size at the program's compile-time.

Note, I am not saying that a slow prefix, like packed in that it can be ignored by an implementor, is not useful. It could be very useful, particularly in computers with multi-level memories such as CDC's ECS, to be able to declare an array as slow. The Elliott 503 of long ago did this very successfully in its Algol. What I am saying is that a slow array is not a random access file. Far from it.

No, a random-access file may be one written by a program which does not know its length until it has been written; for example the generated code file of the B6700 PASCAL. Largely this is written sequentially, but it is tree-structured internally and the compiler needs to make some random accesses to patch up pieces of it. Even more so, a program which accesses an already written random-access file may not know its length. Random accessing a file is a property of the access, not of the file. B6700's have files which may be accessed either sequentially or randomly as you choose (if it is a disk file of course). My suggestion for this is to attach the random access key to the read and write statements, or as Wirth suggests for CDC segmented files, to versions of reset and rewrite. Possibly with an array-connotation syntax:

```
        seek(file1[index])
or      seek(file1,index)
```

(4) What relation is there between PASCAL files and our operating system files?

It is possible to argue that current operating systems support things they call 'files' which are often a mess, and that PASCAL files should have no truck with any of this mess. This is a defensible argument, and I cannot argue against it. If accepted however, it has the effect of relegating PASCAL to the role of an academic language - having an effect on teaching and the future evolution of languages but none on the real world out there. The facts are that real-world files exist; their facilities cannot be completely ignored except at the cost of making the language irrelevant to systems and applications programmers. Some of you may be satisfied with that, but I am not.

What we need rather is to assimilate what is good in real file structure into a pseudo-standard: a document describing preferred extensions to PASCAL. Then implementors would have some idea of what might be a recognized extension compatible with some people, rather than the mixture of suggestions that have been put to me.

SUMMARY

PASCAL has much more to fear from its friends than its enemies. Its two greatest dangers are from naive extensions and PASCAL-fanaticism. The language has defects; it has strengths. Let's be a bit more cautious.

I'd also like everyone thinking about files in PASCAL to ask themselves which of the following sorts of files they are thinking about:

magnetic tape files,

disk files,

printer spool files,
directly attached printers,
files attached to interactive terminals,
card reader files,
and so on.

I am interested of course in the purpose and lifetime activities of such file types, not whether they actually reside on a spinning magnetic thing of 21 surfaces or whatever... The differences in activities are still surprizingly large, and important.

And finally, let me exhort all implementors and users to regard the standard usage of PASCAL files as being limited to their declaration as types and corresponding var objects; their use as var parameters to procedures and functions; and their use with the file buffer and the I/O procedures.

Further, the scope of a file name should be regarded as the scope of its name alone. The question of its lifetime is regrettably one that standard PASCAL does not address adequately.

Prof Arthur Sale
Department of Information Science
University of Tasmania
Hobart, Tasmania 7001

POSTSCRIPT

AN INVITATION TO USERS AND IMPLEMENTORS

This is an invitation to users and implementors of the many PASCALs there are around (though I have little faith in the response ability of implementors) to write to me to say what their PASCAL compiler actually does implement with respect to files. Does it permit file assignment? files as procedure parameters? files of arrays? arrays of files? If I receive anything, and if it permits of a summary, I'll try to write one for a future newsletter. The Revised Report is hardly a guide at all in this area.

Department of Information Science
1977 February 14

LETTER TO THE EDITOR,
P.U.G.N.

Dear Andy,

Three criticisms, I regret to say.

1. PUGN DISTRIBUTION

I understand that you have decided to post overseas subscribers their newsletters by surface mail (other than USA and UK). *I protest vigourously.* Do you realize that with all the ships involved this means I get the newsletter about 3 months after it has been published? Only by courtesy of Judy Mullins have I received a copy of newsletter #7 yet, and when my own copy finally arrives it will be far too late to comment on anything in it, or indeed to carry out any meaningful correspondence. Airmail is a must for post to Tasmania.

2. EDITORIAL SNIPING

In your editorial in Newsletter #7, you took me to task for "wholeSALE bending of PASCAL" and reminded me of an implementor's responsibility to the user community. May I say that I was surprised since I have not indulged in such destructive bending, nor do I think PASCAL will bridge Burroughs users onto other machines. However, what I should especially like to point out to you is that if you are going to object to something, you ought to be specific in your objections. I have no reply at the moment, except to think that you have confused language criticism and insights with implementation intent or fact (on a document which has now served its purpose), or to think you place an inflated worth on some very minor points.

If I may, I'll make two points to illustrate. The first relates to the responsibility of implementors to the user community. I am well aware of this responsibility, and indeed one of the aims of the B6700/B7700 compiler is to be a more searching test of "standards-compatibility" than the CDC compiler

UNIVERSITY OF MINNESOTA
TWIN CITIES

University Computer Center
227 Experimental Engineering Building
Minneapolis, Minnesota 55455
(612) 373-4360

April 26, 1977

Prof. Arthur Sale
Department of Information Science
The University of Tasmania
Box 252C G.P.O.
Hobart 7001 Tasmania
Australia

Dear Arthur,

It's been too long since I have written you a letter. I received your nice personal reply and all the enclosures of March 18 to my personal letter to you dated March 10. Yesterday we got your Burroughs 6700 Status Report. Thanks!

Since returning from Southampton on March 28 I've been swamped with work. On March 31 Niklaus came to the university here to give a talk on Modula (250 persons plus, standing room only). Next week for 4 days, the CDC annual user's meeting was held here in Minneapolis. I began to go through a 40cm mountain of mail and process over 150 new PUG memberships. We started to put together PUGN#8 on April 16. Now we are finishing that up and should go to press within a few days.

Again I'd like to apologize for singling you out as an example in two consecutive editorials. The last exchange of letters has I hope helped me understand that your attitude that I at first perceived as "very opinionated" and "know it all" is actually intended to provoke debate, to prevent dogmatic thinking among over-enthusiastic Pascalers, and to overcome great distances from Tasmania. In short, as the cliche goes, we need people like you. And your valuable contributions to PUG and the Newsletter justify mailing the newsletter to you by airmail and at a loss. But I would like to take you up on your choice of being a distribution center (and perhaps money collector) for Australasia for the next academic year (beginning with #9). We'll have to work out the details this summer (winter!). We have 8 members in Japan, but I suppose Japan cannot be mailed to cheaply from Australia?

On to another topic. When I looked at Judy's letters I discovered that there was one from her to you I'd never seen that would have really clarified the exchange in PUGN#6. It explained their proposed ICL ASCII subset character set. In fact they have taken up the idea proposed by yourself and others which is to process both ASCII and EBCDIC internally as compile options.

On the question of files and the program heading and the larger accusation that Pascal is biased toward CDC computer systems, I'd like to say that I believe:

1) Files as a data structure (sequential access) are a useful concept and therefore files can be special entities represented by variables and used for performing I/O. I don't believe in file assignment though.

2) Arrays are a random-access structure in Pascal and so "virtual" (or slow) arrays would be appropriate for "direct access secondary storage" (read: operating system random-access files). And so arrays can be used for I/O.

3) The program heading is not a "CDC quirk." The first Pascal compilers for CDC machines did not have them; it is not a necessity. CDC Fortran coincidentally has a similar construct. But when you think about it, the program heading would be the ideal place for putting all your computer system dependent information about file attributes (KIND, MAXRECSIZE, etc.) on the B6700 instead of the var declaration. The program heading is a natural way to interface a program to its surrounding environment with formal parameters.

4) Other complaints against CDC bias probably should be rephrased as simple-architecture/multi-register machine bias. Wirth designed Pascal to be run efficiently on today's machines (1970-72) and he has had at least IBM 360, CDC 6000, and PDP-11 experience. So we witness that highly structured computers (such as the B6700 and ICL upper 2900) are among the last to have Pascal compilers. (The problems facing 360 implementations are probably due to interfacing with their dinosaur operating systems). Nagel's DEC-10 compiler, Mike Ball's Univac compiler, and Hikita's Hitac 8000 (Amdahl 470) compiler have shown that "CDC bias" is a phony issue.

I'm glad you are willing to change your views as you indicated regarding syntax. As I said, I'm still learning about the issues myself and have made mistakes and changed my views.

Regarding Pascal's viability and keeping it in the greenhouse, I should say that for better or worse some smaller U.S. computer companies are jumping the gun and have stolen Pascal from the greenhouse. Do not underestimate the "real world" interest in Pascal in the U.S. The PUG membership in the U.S. is at least 40% non-academic versus less than 10% outside the U.S. How about that? Maybe that's why your viewpoint differs from mine. I don't really think I'm ahead of time because one can't control what everyone else is doing. Sure, it would be nice to have consolidation. But just the fact that PUG and PUGN exist have put activities out of reach by spreading the word very widely. If you fear irreparable harm - it has probably already happened - but realistically we couldn't have on the one hand protected Pascal in the greenhouse, and at the same time organized a group for consolidation. We organized openly and, among other things, that's how you and I came to know each other! But you are right about: "if I as a well disposed friend of Pascal can find holes, be assured that real enemies will be less forgiving." I'm hoping the news #8 will bring regarding standards will be encouraging news to you, and I apologize that I can't fit it here in this letter.

Minnesota usage of Pascal? I did point out that the #7 editorial did not say that Pascal meets Waite's criteria, but rather in trying to spread Pascal usage at Minnesota, Waite's guidelines proved to be very useful in practice. You wanted a breakdown on usage:

Processor	number of research and production runs/number of instructional runs (1st 9 months 76-77)	75-76	74-75	/1st 9 mo76-77	75-76	74-75
Pascal	(28,537)	38,159	13,968	(77,604)	65,158	8,928
MNF Fortran	(239,032)	417,317	225,501	(327,473)	393,107	526,252
Cobol	(44,750)	42,756	33,711	(1,762)	6,250	8,555
APL	(4,312)	673	605	(3,935)	6,262	6,658
SNOBOL	(8,715)	11,645	5,505	(23,271)	28,727	36,494
SIMULA	(686)	2,192	2,310	-	-	-
ALGOL	(615)	1,763	2,581	(75)	1,373	1,998
BASIC	(3,412)	28,103	445	(145,977)	448,814	1,476,984

(*Other processors include COMPASS(assembler), DARE,EMULATE,GPSS,LISP,MIMIC,MIXAL, PL/1, SIMSCRIPT, RPG. There are over 100 interactive terminals for student use; the University of Minnesota has 55000 students on the Twin Cities campus.*)

Regarding the printing of your and Judy's correspondence, that's fine (except that space may not permit). As will be evident in #8, there is more than enough debate going right now. Your implementation notes, etc. are very nice. We'll print most of them in #9. Thank you very much for being understanding. I've resolved to be more careful in the future.

Sincerely,

Andy

for example. I greatly regret that the existing PASCAL user-community does not have much of a clue about standardization; most seem to think that the CDC compiler defines the standard! There are a number of other important goals too; I intend this to be much more than the usual PASCAL 'toy' compiler.

You make one good point (which I think you cannot have meant). Why not stick to Burroughs Algol? I could say why not stick to FORTRAN too, but you'd probably object to that. In fact, I believe PASCAL's ecological viability when compared to Burroughs Algol or standard FORTRAN is very dubious at the moment, but I'll treat of that later. The important thing to realize is that Burroughs compilers are good (really good) and PASCAL's viability in Burroughs must rest on real strengths, not just claims. This, coupled with the known weird features of CDC systems (and thence PASCAL) must lead to the uncovery of unfortunate aspects of PASCAL. I cannot help it.

3. PASCAL SUPPORT

I was surprised to see you write in the editorial that you believed PASCAL meets Bill Waite's criteria for ecological viability, for my impression is quite the reverse. Possibly in CDC machines it might have enough support, but that is a tiny fraction of the computing community. To take some examples, I have assiduously tried to amass the PASCAL software that PUGN assures me is around. The results have been decidely poor. Apart from interchange media problems, most programs contain machine dependencies of considerable subtlety, and totally inadequate commentary. Not all, but most. The original XREF used at least seven non-standard features which had to be repaired, some with difficulty, and even then its specifications left a lot to be desired. To my knowledge, no available cross-referencer is able to distinguish between names which are lexically the same but declared at different levels, nor can they cope with long names (say 72 characters?).

To summarize, I think your editorial is ahead of time. We certainly don't need crusaders yet, we need some consolidation before irreparable harm is done. At its present state of development, PASCAL stands to go under the FORTRAN steamroller, for precisely Waite's reasons. And really, what do you mean when you say that PASCAL is the third out of 20 languages in four years? In Minnesota? Measured by what? If I was advising someone to choose a language to write a significant numerical piece of software at this point in time, I couldn't (regretfully) advise them to use PASCAL. It would be irresponsible.

So much for the criticisms. Can I still assure you that despite the bits of rubbish here and there in the Newsletter, it serves a very useful purpose. I'll keep on contributing because this is a critical point in PASCAL's development, and because its well WIRTHwhile. Without the newsletter, wide communication would be much more difficult, and your policy of no censorship or refereeing is conducive to good development.

I'd like, too to put a question to you. In Newsletter #6, you published some of the correspondence between Judy Mullins and myself on implementations on Burroughs B6700s and ICL 2900s (which we sent you). We've been carrying on an active correspondence, some of which would be of interest to the PASCAL community. In some ways though, such a practice could be misunderstood or embarassing as half-baked ideas come to light, if it were all reprinted by you. The important issues often get rewritten as notes to PUGN (some examples may get into #8), but what I ask is this: would it be useful for the readership to look over the letters as they develop? I'm game, and I'll ask Judy, but I am uncertain as to the merit of the practice. What do you think?

Yours sincerely,

Arthur Sale
Professor of Information Science
University of Tasmania
(Burroughs B6700 implementor)

PS. By the way, apropos of your plea for help, if there is anything doing on standardization, you can count me in. I've had a fair bit of experience with standards and standards committees, and I know just how large a task there is to do. Perhaps I can help as co-ordinator?

UNIVERSITÉ DE NICE
LABORATOIRE D'INFORMATIQUE
PARC VALROSE
06034 NICE CEDEX
TÉL. 51 91 00

Nice, le 4th March 1977

Dear Andy :

I am sorry that the paper I promised to write is so late, but you are preparing PUG Newsletters faster than I can read them. At last, here is the paper on the Pascal implementation we are developing for the CII Iris 50. It is not yet in the form requested by Tim Bonham, because nothing is terminated, and anyway the Iris 50 is a machine which does not exist in many copies.

A Pascal subgroup has been officially set up as a part of the group "Languages and programming" of AFCET. To give comparisons, and with the corresponding scale changes, AFCET is the French counter-part of ACM, and the group on languages and programming is something like SIGPLAN, so this Pascal subgroup is something like STAPL within SIGPLAN within ACM; very complicated indeed. The first meeting of the group will take place in Nice on June 13 and 14. A newsletter is planned to begin at the end of the present month. Answers to a questionnaire show strong interest of participants on frequent information exchanges, and desire to keep good bonds with PUG. If there is no copyright problem, and with your authorization, I intend to extract some most important informations from PUGN for our newsletter, and even maybe to directly copy some pages.

Do you think it would be interesting to publish some brief information about my compiler writing system (written in Pascal and generating compilers written in Pascal) in a section of PUGN about software writing tools? This system is probably bigger than ordinary tools (about 6000 Pascal lines), and has a very special purpose, but it presents some interest for the community.

I am sorry to have given an erroneous information in my preceding letter. No Pascal compilers for the IBM 1130 were made in Neuchâtel. A Pascal-S compiler (not an interpretor) has been made by Helmut Sandmayr Neu Technikum, CH-9470 Buchs, Switzerland. I apologize for the error to people who have already written to Neuchâtel.

Yours sincerely,

O. LECARME

University College, Cardiff

Professor R. F. Churchhouse, B.Sc., M.A., Ph.D., F.B.C.S.
Head of Department of Computing Mathematics
Mathematics Institute, Senghennydd Road, Cardiff
Telephone Cardiff 44211 Ext. 2677 & 2678

28th March 1977.

Dear Andy,

Following the recent PASCAL Symposium at Southampton may I make an impassioned plea on behalf of potential future users of the language.

So many people talk glibly about not re-inventing the wheel. Yet as I survey the many and diverse efforts at implementing PASCAL on mini/micro-computers (particularly PDP-11s) surely this is what we are in danger of doing. For unless we both a). recognise the value to others of the software products we originate and b). invest accordingly in faithful standardisation, intentional portability and quality documentation, much is vanity. To take the specific example of providing PASCAL for student teaching purposes on a PDP-11, what is the use of existing "implementations" which a). their originators have never even thought of as potentially useful to others and b). are non-standard, tied to a particular operating system without provision for change, and atrociously written up? My plea is to all good PASCALlers to honour the original spirit of the language by practising these principles and, possibly much more important, doing their utmost to persuade others to do so also. Down with back-street implementors!

Yours sincerely,

Nick Fiddian

IMPLEMENTATION NOTES

IMPLEMENTATION CHECKLIST

IMPORTANT ! ! !

We have added one new item to the Implementation Checklist (reprinted below) to indicate the kinds of library support provided by implementations. Once again we must ask implementors to follow the Checklist, and to submit notices in "camera-ready" form. Because of the large number of implementations being reported, we request that all notices be single spaced.

1. Names, addresses, and phone numbers of implementors and distributors.
2. Machine(s) (manufacturer, model/series).
3. Operating system(s), minimal hardware configuration, etc.
4. Method of distribution (cost, magnetic tape formats, etc.).
5. Documentation available (machine retrievable, in form of a supplement to the book: Pascal User Manual and Report).
6. Maintenance policy (for how long, future development plans, accept bug reports).
7. Fully implements Standard Pascal (Why not? what is different?).
8. Compiler or interpreter? (written in what language, length in source lines, compiler or interpreter size in words or bytes, compilation speed in characters per second, compilation/execution speed compared to other language processors (e.g., FORTRAN)).
9. Reliability of compiler or interpreter (poor, moderate, good, excellent?).
10. Method of development (from Pascal-P, hand coded from scratch, bootstrapped, cross-compiled, etc.; effort to implement in person months, experience of implementors).

NEW!! 11. Are libraries of subprograms available? Are facilities for external and FORTRAN (or other languages) procedures available? Is separate compilation available?

GENERAL INFORMATION (77/4/28).

As an aid to persons searching for implementations, an index to the Implementation Notes section for Newsletter issues 5 through 8 is printed at the end of this issue. Unfortunately, we had to leave out or summarize a number of letters and notices because of space constraints.

-Jim Miner

All Implementors:

Why not use the Pascal Newsletter to help yourselves (and all of us) disseminate news of new releases for existing implementations to all the sites on your distribution list? Also, to ensure that everyone on your list receives the Newsletter (and is a member of PUG) please send out an All Purpose Coupon with each copy of your implementation that you distribute.

Comment on Micro-processors.

One of the more interesting developments that we have seen is the increasing use of Pascal as a micro-computer programming language. Among these machines we count DEC's LSI-11, the Intel 8080, the Motorola 6800, TI's 9900, and the Zilog Z-80. (I'm just not sure about the Nanodata QM-1....) Most of these are interpreted, but native code implementations are beginning to appear (see Pete Zechmeister's Intel 8080 notice in this issue).

Another fascinating rumor, which was published in two places (Byte, and Computer Faire) suggests that the next Zilog processor will be based on Pascal -- with the instruction set including some Pascal-like constructs. Apparently users and designers are beginning to see the advantages of a simple yet powerful language. Perhaps the experience will lead to cleaner micro architecture.

SOFTWARE WRITING TOOLS

Responding to the call for a central clearinghouse for software writing tools, Richard J. Cichelli has volunteered to distribute them and will announce a formal policy in Newsletter #9. At our suggestion Rich will limit distribution to implementors who distribute Pascal systems and who will include the software tools in each distributed copy. This is to prevent an absurd workload for Rich. Rich is probably in possession of the largest number of software writing tools in Pascal and for Pascal programmers. (See the article entitled "Pascal Potpourri" in Newsletter #6.)

PASCAL-P

Remember, there is a policy of no maintenance promised on Pascal P4. It is the final version from Zurich. Nevertheless, Christian Jacobi (ETH, Zurich) has provided us with two sets of changes (printed below) to be made to version P4, mainly correcting bugs in address calculations and code generation. Note that the form "name.number" refers to the sequencing on the compiler source as distributed.

Unfortunately we have not received the results of the Pascal-P questionnaire which appeared in Newsletter #5. Chris informed us on February 14 that the results were in preparation.

UPDATE 1 to Pascal P4 January 1977

Replace line BOOT.4 by

```
      for i := ordminchar to ordmaxchar do sop[chr(i)] := noop;
```

Replace line P.477 by

```
          load; genlabel(lcix);
```

Insert after line P.479

```
          genujpxjp(57(*ujp*),lcix);
```

Replace line P.147 by

```
                        begin align(lspi,displ);
```

Replace line P.424 by

```
                                          locpar := locpar+ptrsize;
                                          align(parmptr,locpar);
```

Insert after line PASCP.3200

```
                align(parmptr,llc1);
```

Replace line P.531 by

```
                    if idtype↑.form > power then
```

PASCAL TRUNK COMPILER

Dear Mr. Mickel,

I send you here the information about the trunk compiler you asked for:

1. Implementation + distribution
   H.H. Nägeli
   Institut für Informatik
   ETH-Zentrum
   CH-8092 Zürich / Switzerland
   Tel. 32 62 11
2. The trunk compiler is the machine independent part of a Pascal compiler in which the code generation has to be inserted.
3. -
4. Distribution on magnetic tapes. Costs SFr. 50.-- .
5. Documentation (in German) will be available in May 77.
6. Maintenance policy: no policy defined yet.
7. Full Pascal is treated.
8. The trunk compiler is a Pascal program with a certain number of empty procedures.
9. Reliability: moderate.
10. Development: from Ammann's Pascal CDC 6000 compiler.

Sincerely yours,

H.H. Nägeli March 3, 1977

PASCAL J

Manpower problems have forced us to cancel the projected February Release of PASCALJ. Although we have made some progress in our efforts to improve the bootstrapping process, we lack the supporting documentation necessary for a distributable product. We will therefore continue to distribute the September 1976 version of the system to those requesting it.

We would like to emphasize once again that we consider the portability of this version to be inadequate, with implementation times ranging upward from six man-months required. Reduction of this implementation time is our prime concern, and is absorbing the meagre resources which are currently available to the project. As soon as significant progress has been made in this direction we shall release a new version. In the meantime, we shall attempt to fix any reported errors.

- Software Engineering Group

UNIVERSITY OF COLORADO
DEPARTMENT OF ELECTRICAL ENGINEERING
BOULDER, COLORADO 80309

MODULA

Niklaus Wirth has published three articles describing his latest language which he calls Modula. The articles appear in the March, 1977, issue of Software - Practice and Experience (vol. 7). It is our policy to discuss languages adhering to the principles embodied in Pascal, and some of the characteristics of Modula make it a very attractive programming tool, particularly for small, peripheral oriented machines. For this reason we reprint here the Summaries (abstracts) of the articles. Please note that Niklaus considers Modula still in the experimental stage and the Zurich implementation is not for distribution.

"Modula: a Language for Modular Multiprogramming", S-P&E 7 (1977), pages 3-35.
SUMMARY
"This paper defines a language called Modula, which is intended primarily for programming dedicated computer systems, including process control systems on smaller machines. The language is largely Pascal, but in addition to conventional block structure it introduces a so-called module structure. A module is a set of procedures, data types and variables, where the programmer has precise control over the names that are imported from and exported to the environment. Modula includes general multiprocessing facilities, namely processes, interface modules and signals. It also allows the specification of facilities that represent a computer's specific peripheral devices. Those given in this paper pertain to the PDP-11."

"The Use of Modula", S-P&E 7 (1977), pages 37-65.
SUMMARY
"Three sample programs are developed and explained with the purpose of demonstating the use of the programming language Modula. The examples concentrate on the uses of modules, concurrent processes and synchronizing signals. In particular, they all focus on the problems of operating peripheral devices. The concurrency of their driver processes has to occur in real time. The devices include a typewriter, a card reader, a line printer, a disk, a terminal with tape cassettes and a graphical display unit. The three programs are listed in full."

"Design and Implementation of Modula", S-P&E 7,67-84 (1977)
SUMMARY
"This paper gives an account of some design decisions made during the development of the programming language Modula. It explains the essential characteristics of its implementation on the PDP-11 computer, in particular its run-time administration of processes and the mechanism of signalling. The paper ends with some comments on the suitability of the PDP-11 for this high-level multiprogramming language."

FEATURE IMPLEMENTATION NOTES

READING AND WRITING SCALARS

Introduction

It has long been a source of irritation that "standard" PASCAL does not permit the reading of boolean values (though it permits their writing), and does not permit either reading or writing of programmer-defined scalar types. In Burroughs B6700/B7700 PASCAL, both these deficiencies are remedied, and the regularity of PASCAL is improved. The utility of this step should not need labouring, especially as it dispenses with unnecessary rules, and in view of its obvious uses in an interactive environment.

Insert after line PASCP.3204

```
                    if vkind = actual then
                    begin
```

Insert after line PASCP.3207

```
                    end;
```

Corrections to the Pascal P4 System UPDATE 2

With kind regards

Ch. Jacobi

Ch. Jacobi

Replace line p.122

```
    flc := l+k-(k+1) mod k
```

Replace line p.528

```
      cstptrix := 0;
      topnew   := lcaftermarkstack;
      topmax   := lcaftermarkstack;
```

The first correction delivers an improvement of storage allocation in case flc = 0 (e.g. records).

The second correction is evident.

IMPLEMENTATION NOTES

Craig E. Bridge (DuPont, Wilmington, Delaware) furnished the modifications printed below to allow the compiler to be cross-compiled between machines with different character sets. He also notes in a letter dated Feb. 16, 1977, (which was not printed for lack of space) that where cross-compilation is to be done very often the cross-compiler should be modified to generate proper code (jump table) for statements of the form "case chartype of ... end".

```
*IDENT DUPONT
*DECK PASCP
*/    DUPONT MOD SET FOR PASCP VERSION P4 06-JAN-77    C.E. BRIDGE
*/
*/        ELIMINATE LAST HOST MACHINE CHARACTER SET DEPENDENCY THAT
*/        PROPAGATES FROM THE HOST COMPUTER DURING CROSS CODE GENERATION.
*/
*/        NOTE: THE PASCP COMPILER ALREADY HAD A UNIVERSAL INPUT PROCESSOR
*/        HOWEVER THE CASE STATEMENT CODE GENERATION PATTERN BANKS ON THE
*/        ORDINALS OF SETS (INCLUDING THE IMPLIED CHARACTER SET) TO BE
*/        THE SAME ON THE HOST AS IT IS ON THE TAGET MACHINE.  THIS IS
*/        NOT NECESSARILY TRUE OF CHARACTER SETS.
*/
*/        IN PARTICULAR,     CASE CH OF ..:..    GENERATED A JUMP TABLE
*/        USING THE ORDINATES OF THE HOST MACHINE CHARACTER SET.  SEE
*/        STATEMENT PASCP.376
*/
*/        FURTHERMORE, ANY PASCAL PROGRAMS WITH STATEMENTS OF THE
*/        ABOVE FORM CANNOT BE CROSS COMPILED FOR MACHINES WITH DIFFERENT
*/        CHARACTER SETS UNLESS THE CASE STATEMENT CODE GENERATION
*/        PATTERN IS MODIFIED.
*/
*/        NOTE:  _ IS AS CLOSE TO THREE HORIZONTAL BARS (CDC DISPLAY CODE
*/        60 OCTAL AND EXTERNAL BCD 36 OCTAL AS OUR FONT CAN COME.
*/
*/       ***************************************************************
*/       *WARNING:  THIS MODSET IS UNTESTED.  WE DONT HAVE A MODIFY      *
*/       *PROCESSOR AVAILABLE ON SITE.  THE MODS WERE MADE USING         *
*/       *A TEXT EDITOR AND THEY APPEAR TO WORK.  PLEASE INSPECT         *
*/       *THE RESULTS BEFORE DISTRIBUTING BEYOND YOUR SITE.  USER BEWARE.*
*/       ***************************************************************

*DELETE P.57
      CHTP = (LETTER,NUMBER,SPECIAL,ILLEGAL,CHSTRQUO,CHPERIOD,CHLT,
              CHGT,CHLPAREN,CHSPACE);
*DELETE PASCP.376 PASCP.379
     CASE CHATP[CH] OF
       LETTER:
*DELETE P.79
            UNTIL CHARTP[CH] IN [SPECIAL,ILLEGAL,CHSTRQUO,CHCOLON,
                                 CHPERIOD,CHLT,CHGT,CHLPAREN,CHSPACE];
*DELETE PASCP.395
       NUMBER:
*DELETE PASCP.453
       CHSTRQUO:
*DELETE PASCP.474
       CHCOLON:
*DELETE PASCP.480
       CHPERIOD:
*DELETE PASCP.486
       CHLT:
*DELETE PASCP.495
       CHGT:
*DELETE PASCP.501
       CHLPAREN:
*DELETE PASCP.514 PASCP.516
       SPECIAL:
*DELETE P.85
       CHSPACE: SY := OTHERSY
*DELETE PASCP.453
       CHSTRQUO:
*DELETE PASCP.474
       CHCOLON:
*DELETE PASCP.480
       CHPERIOD:
*DELETE PASCP.486
       CHLT:
*DELETE PASCP.495
       CHGT:
*DELETE PASCP.501
       CHLPAREN
*DELETE PASCP.514 PASCP.516
       SPECIAL:
*DELETE P.85
       CHSPACE: SY := OTHERSY
*DELETE P.591
       CHARTPC_/_] := SPECIAL; CHARTPC_(_] := CHLPAREN;
*DELETE P.593 P.596
       CHARTPC_=_] := SPECIAL; CHARTPC_ _] := CHSPACE;
       CHARTPC_,_] := SPECIAL; CHARTPC_._] := CHPERIOD;
       CHARTPC_'_] := CHSTRQUO; CHARTPC_[_] := SPECIAL;
       CHARTPC_]_] := SPECIAL; CHARTPC_:_] := CHCOLON;
*DELETE P.598
       CHARTPC_<_] := CHLT; CHARTPC_>_] := CHGT;
```

Example

```
program example(output,input);
    type
        answers = (yes,no,maybe);
    var
        reply   : answers;
begin
    read(reply);
    write(output,reply);
    writeln(reply:6);
end.
```

Semantics of reading

The input stream is scanned for an alphabetic character. It and succeeding alphanumeric characters are assembled into a "lexical token" according to PASCAL rules, and then compared with a stored table of the programmer-defined constant-names of the type. If a match occurs, the appropriate constant value is stored into the variable in the read list, otherwise a read error occurs. The construction of the "lexical token" is terminated by any character which is not alphanumeric (usually a space or a comma).

Semantics of writing

The characters of the constant-name corresponding to the scalar value, preceded by a single space, are inserted into the output stream if no field width is specified. If a width is specified, the name is inserted into a character field of that size, right justified and filled with preceding spaces if necessary. If the name will not fit in the field, or if the scalar-value is somehow out-of-range, a non-fatal write error occurs.

Boolean values

Values of boolean type are treated exactly as if declared:

```
type
    boolean = (false,true);
```

and thus the external representation of any boolean value is *false/true* (and not *F/T*, or *0/1*).

Burroughs B6700 compiler features

Since the B6700 compiler is a true anylength identifier system, all characters of the constant-names and of the input tokens are significant in distinguishing one name from another. In addition, since lower-case letters are permitted, the letters in input tokens are upper-cased before comparison with the stored name-table which is stored in canonical upper-case form by the compiler. Persons requiring their programs to be portable should be aware that "standard" PASCAL permits implementors to ignore names after the first 8 characters, though this feature is not "standard".

Scalar name tables

The name-table is not created by the compiler unless the compiled program contains a read or write with a scalar element. The table only includes the types the compiler finds are necessary (except for boolean, which is handled by a table internal to the read/write intrinsic procedure). The run-time space penalty is typically very small.

> RECOMMENDATION
>
> *If this increased regularity is attractive to an implementor of PASCAL, or if a teacher can convince an implementor to include it, I suggest adherence to the above ideas as far as possible. This applies to the reading of boolean values alone, as well as to a more comprhensive adoption of the facility.*

POINTER VALUES

Introduction

This implementation note serves to document some relevant decisions relating to the representation of values of a *pointer* type in B6700/B7700 PASCAL. The note may be useful to users of this computer and to other implementors.

Normal pointer values

The representation of pointers in the B6700 and B7700 computers could have been a problem of considerable difficulty (perhaps impossible) if PASCAL had been defined so as to allow pointers to objects outside its heap. Since it did not permit this, it allowed the heap to be implemented as a single segment of virtual storage (paged into 256-word pages). Normal pointer values are thus represented as integer words, being utilized as subscripts into the heap vector when a pointer access is required. It is important to realize that the concept *address* does not exist in Burroughs B6700/B7700 computers.

The legal values of a pointer variable range from zero to an upper limit which is compiled into a PASCAL program. The default limit gives 1000 words, but this may be set at any value by the compiler option HEAP.

The *nil* value

The *nil* value, which points nowhere, is implemented as a very large numeric value. Any reference to an apparent object (even if it includes record selection or array indexing) through a pointer which has this value will cause a machine interrupt when the access is attempted (because the subscript is out-of-bounds of the heap size shown in the heap descriptor). This check has no speed penalty as it is carried out by the hardware. It remains possible to compare pointers for equality even with the *nil* value.

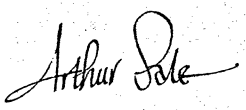
The uninitialized value

The value of a pointer variable before it is first specifically defined by an assignment, read, or whatever, is left to the implementor's discretion by "standard" PASCAL. It is worthwhile pointing out here that the uninitialized value may perhaps not be best implemented by *nil* and a special representation should be considered (though on some computers there may be no other suitable value).

Because of the importance of pointers, and the responsibility of compilers to detect as many illegal constructs as possible (as well as correctly compiling the correct ones), the uninitialized value for pointers in the Burroughs B6700/B7700 PASCAL is not zero (the B6700 norm), nor is it *nil*. Uninitialized pointers are set to B6700 words with a tag of six. Such tag-six words in the B6700 and B7700 computers can be overwritten with a numeric or other-type operand (tags 0, 2 & 4), but an attempt to utilize a tag-six word in arithmetic or indexing is illegal and causes a machine interrupt. The use of an uninitialized will therefore be detected (whether in a comparison or an access) and will cause program termination.

Conclusion

The B6700/B7700 PASCAL compiler applies stringent testing to PASCAL pointer values so as to enforce compliance with the "standard". Implementors on other computers may wish to consider whether they can make effective use of the *nil* value, and of the difference between *nil* and the uninitialized value.



Arthur Sale
Professor of Information Science
University of Tasmania
(Burroughs B6700 implementor)

1977 February 15

Implementation Note on Run-time Pointer Tests.

The paper by Charles Fischer and Richard LeBlanc described in the Here and There (Articles) section presents a method for feasibly implementing run-time pointer checks. The method has been installed successfully on their Univac 1100 compiler, as well as by John P. Strait on the CDC-6000 compiler and by John Reynolds on the ICL 2970 compiler.

Simply stated, each unique element allocated on the heap is assigned a unique integer or "key" (a counter starting at 1) which is stored with the pointer variable and with the heap element. The key and pointer value (address) are transmitted together during pointer assignment or parameter passing. A pointer reference is considered valid only if tne key in the reference and the key in the heap element match (comparison of keys for equality). Therefore, "dangling references" to a heap element which has been disposed will be detected (implying that DISPOSE changes the key in the disposed element). Note that the method is nearly secure -- it is possible (but very unlikely) that a key will match with garbage on the heap existing in the place of a disposed element. Similarly, undefined pointers will have undefined keys which could, with low probability, match their referent keys.

-Andy Mickel

MACHINE DEPENDENT IMPLEMENTATIONS

Burroughs 3700, 4700

Dear Tim:
Here is a brief outline of our Pascal project; please note that although our intentions were to produce both B4700 and B6700 implementations, the latter has not been possible. Fortunately, Professor Sale is producing a B6700 compiler.

1. Implementors.

R. M. Lansford
3620 Greenhill Rd.
Pasadena, Ca. 91107

P. L. McCullough
110 S. El Nido St.
Pasadena, Ca. 91107

W. C. Price
480 Pembrook Dr.
Pasadena, Ca. 91107

2. Environment.

This implementation will run on Burroughs B37/4700 machines, with Accumulator operators, under MCPV 5.7 and the Time-Sharing System.

3. Distribution.

No plans at present - the need has not arisen.

4. Documentation.

What there is exists as a forward to the program listing, in the form of a supplement to the Pascal User Manual and Report.

5. Maintenance policy.

None. Development has terminated. "If you find'em, fix'em."

6. Unimplemented features.

a. real arithmetic
b. formal procedures and functions
c. files, with the exception of the text files Input and Output.

7. Added features.

a. segmentation
b. symbolic procedure call tracing
c. stack checking and statistics.
d. packing is automatic

8. Compiler development.

The compiler was bootstrapped from an early P1 compiler obtained from Cal Tech.

The compiler consists of two passes. The first is written in Pascal and emits augmented P-code. The second pass (written in BPL, a PL360-like assembler), generates 4700 code from the P-code.

The first version of the code generator was written by Mike Mahon in 2 man-months. An additional 8 man-months have been expended in teaching the compiler about such things as optimal variable size and alignment, segmentation, etc.

The results are:
Pass 1 : 4000 lines of Pascal, compiled @1000 lines/min.
Pass 2 : 2500 lines of BPL, taking 45 secs to generate code for Pass 1 of the compiler.
110K bytes are needed for a logical (reasonable) segmentation of the compiler.

9. Compiler reliability.

Good, but not excellent.

Sincerely,

R. M. Lansford

R. M. Lansford January 17, 1977

BURROUGHS 6700

Antti Salava (Department of Computer Science, University of Helsinki, Toolonkatu 11, SF-00100 Helsinki 10, Finland) reports "we have here an almost-finished Pascal compiler running on the B6700. The compiler is written in Burroughs Extended Algol and generates B6700 machine code. I won't go to details now because we are currently preparing a report on our Pascal implementation."

(* Received 77/1/17. *)

Ken Bowles reports that a B6700 implementation exists at the University of California San Diego. The implementation was evolved from Pascal P2 by Mark Overgaard and Jim Madden (cf. Pascal Newsletter #4). The latest version is a real compiler, written in Pascal, which produces native code for the B6700. Current compile speed is 5000 lines per minute, but expected improvements could make that 10000 lines per minute -- as fast as the Burroughs Fast Algol compiler. Virtually all of the Burroughs I/O facilities are supported. Distribution is scheduled to start in mid-summer. For more information, contact Henry Fischer, UCSD Computer Center, La Jolla, CA 92093 (714/452-4050).

BRIEF NOTES ON A PASCAL IMPLEMENTATION AT OTAGO UNIVERSITY

About 18 months ago we obtained an implementation of PASCAL for a Burroughs B6700 computer from Karlsruhe University. For reference, this compiler produces symbolic code for a hypothetical stack machine. This symbolic code must be assembled to produce absolute machine code which may then be interpreted. Both the assembler and interpreter are written in Burroughs Extended ALGOL. Since the compiler itself is written in PASCAL, the compilation of a program involves the interpretation of the compiler code file. As a consequence, on our machine, it took about 50 minutes processor time to compile the compiler.

To improve efficiency I rewrote the compiler in Extended ALGOL. This version still produces the same symbolic code but is considerably faster. For example the ALGOL version of the compiler takes less than 1 1/2 minutes process time to compile the PASCAL version of the compiler.

I have started work on turning the ALGOL version into a true compiler for the B6700 but priority of other work has caused delays. I will probably be getting down to it in earnest again in about July of this year.

Copies of this compiler have already been sent to Massey University, Palmerston North, New Zealand and to Warwick University in England. If any one else would like a copy, could they send a tape to me and I will return same with all PASCAL material we have, plus brief notes on usage. The tapes can be in one of the following formats (please specify which is required):-

(a) 1600 bpi, Phase encoded, 9 track, B6700 library tape
(b) 800 bpi, NRZ, 9 track, B6700 library tape
(c) 1600 bpi, Phase encoded, 9 track, USASI multi-file tape
(d) 800 bpi, NRZ, 9 track, USASI multi-file tape.

Chris Bishop
Computing Centre
University of Otago
P.O. Box 56
Dunedin
NEW ZEALAND.

PASCAL FOR THE BURROUGHS B6700/B7700 - STATUS REPORT

1977 April 20

Professor A.H.J. Sale
Department of Information Science,
University of Tasmania.

The PASCAL compiler for the Burroughs B6700/B7700 computers has been operational on the University of Tasmania's B6700 installation for approximately two months in normal shift time, and has caused no operational problems at all in that time. It is used by staff, and by students of the Department of Information Science for coursework.

A restricted release has taken place of two B6700 sites in New Zealand to enable the compiler to operate under less favourable conditions than its nursery site, and to elicit comments (favourable or not). No more sites will be supplied the current version so that potential sources of error-reports can be kept to a manageable size, though at present no errors have been reported from any remote site. Work has now started on a second release which will remove three restrictions in the present version which are annoying.

A supplement to the *PASCAL User Manual and Report* has been prepared, and is available to interested persons by writing to the Department. It details the interpretations to be given to undefined areas of the PASCAL documents, cautionary material on non-standard features of other PASCALs, B6700-specific features, and differences from CDC PASCAL-6000. A *Reference Manual* is in preparation (a dictionary-style document), but is not yet complete.

To control error-reporting and the consequent work, we have also adopted a formal approach (more professional perhaps!) which may be of interest to other PASCAL implementors who want their implementation to be kept under control, and more than the usual plaything. Each site supplied with a copy of the compiler is registered with us, and given a supply of FTR-forms (Field Trouble Reports) which are personalized to that site. On detecting an apparent bug in the compiler, a responsible person in the site will complete the FTR-form (numbered), return a copy to us, and wait. Our response is to acknowledge the FTR as soon as possible, indicating our initial assessment of it. If the problem can be detoured (in other words, avoid the problem area), or the compiler or the intrinsics patched, a patch notice is issued: immediately to the reporting site, and in a regular cycle to other sites. All sites will get a regular report on FTRs still extant (not yet finalized), and on the patch notices issued. It is the responsibility of each site to keep the compiler's patch level in the version number corresponding to the latest level. This is printed on each compilation listing for checking purposes.

Examples of the three forms are attached, in case anyone wishes to copy them.

We have not yet formalized the treatment of what might be called *New Feature Requests*. A quite large number of FTRs turn out to be of this kind rather than genuine error situations.

FIELD TROUBLE REPORT : B6700/7700 PASCAL

FTR No PASCAL	XYZ 01	

FTR number

installation code

Installation and address:

Computing Centre,
The XYZ Corporation,
Somewhere,
Australia.

Date of FTR: _____

Person authorizing FTR: _____

Description of problem (if necessary use extra sheets):

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Can the problem be detoured? *Yes:* ☐ *No:* ☐ *Irrelevant:* ☐

(If YES, then attach brief description of detour used.)

Do you want it fixed *immediately:* ☐
soon: ☐
sometime: ☐ ?

Please attach a listing of a small program that exhibits the problem, with supporting information. If the problem cannot be isolated in a small program, be prepared to receive a request for a tape if the problem cannot be otherwise resolved. Do not send a tape unless requested.

ACKNOWLEDGEMENT OF FTR FOR B6700/7700 PASCAL

FTR No PASCAL		

Date received: _____

Date acknowledged: _____

Your FTR is

- ☐ *Ignored. We do not consider it requires action.*
- ☐ *Too hard. We acknowledge your problem, but it is too hard to solve at present.*
- ☐ *Noted. The problem requires further study and we cannot forecast when a solution will be forthcoming.*
- ☐ *In process. We have some idea of the problem, but it will take a few weeks to resolve.*
- ☐ *Patched. The attached patch notice should resolve the problem. The change will be in the next release.*
- ☐ *Already solved. Check the notices you have received as we believe the problem has been reported & solved.*
- ☐ *Misclassified. Your FTR will be treated as a Field Suggestion, rather than a Trouble Report.*
- ☐ *Concurrent. Another FTR has reported this, and will be treated as the reference FTR (No _____).*
- ☐ *Other:* _____

Attached please find:

- ☐ *A patch for the PASCAL compiler source*
- ☐ *A patch for the PASCAL intrinsics source*
- ☐ *A suggested detour around the problem.*
- ☐ *Other material:* _____

Acknowledgement authorized by: _____

The reliability and robustness of the compiler have been excellent. Its performance is similarly good; the execution speed of the compiler being almost identical to the B6700 Algol compiler, and its needs for compilation space being about 50-60% that of the Algol compiler (probably due to PASCAL's lesser complexity). In execution, the compiled PASCAL programs run slightly slower than equivalent Algol programs on average, but the difference is usually within 20% and fairly negligible. Fortran-compiled programs usually execute about 20% slower still unless the vectormode optimization is invoked.

At the risk of sounding repetitious, I would like to re-emphasize the importance of the Waite criteria to which the Editor drew attention. A professional attitude is essential for the success of PASCAL; otherwise we run the risk of yet another fly-by-night language, or almost as bad, having PASCAL's impact confined to educational institutions.

Arthur Sale

PATCH NOTICE FOR B6700/7700 PASCAL

PATCH No PASCAL		

patch number

compiler release number

Date of patch: _____

Person authorizing: _____

Origin of Fault:

- ☐ *FTR No PASCAL-* _____
- ☐ *Internal discovery*

Brief description of fault repaired:

Description of patch:

File name: _____ *Version:* _____

COMPUTER AUTOMATION LSI-2

(* Computer Automation received some attention for their announcements of this implementation which appeared in the trade papers Computerworld (Feb. 7, 1977) and Computer Weekly (Feb. 17, 1977). (Also see the Here and There Applicatons section.) By way of comparison, CA sells their FORTRAN IV for $1600 to $1700, and their operating system for $1900 to $2000. A glance at the LSI-2 Pascal User's Guide shows the following. Only 2 levels of static nesting are allowed (p 2-4). The operators AND, OR, and XOR can be applied to integer as well as Boolean operands. The reserved words FILE, GOTO, LABEL, and PACKED are not supported (p 2-5). Mixed mode arithmetic is not supported (p 2-6). The following standard functions are not supported: ODD, EOLN, EOF, SQR, ROUND, SIN, COS, ARCTAN, LN, EXP, SQRT (p 2-6). *)

ComputerAutomation

March 22, 1977

Dear Andy:

Computer Automation, using Brinch Hansen's Pascal compiler, has implemented Sequential Pascal on its LSI-2, 16-bit minicomputer running under its operating system, OS, configured with 32K memory and moving head or floppy disk.

The Pascal system, released December, 1976, is distributed on floppy disk for a cost of $900.00. Documentation includes the Jensen and Wirth manual and a user's guide explaining the operation of Pascal under Computer Automation's OS. The Pascal compiler is fully supported including acceptance and response to user trouble reports.

The compiler supports Hansen's implementation of Pascal as discussed in #6. However, the I/O capabilities presently are based on the operating system for their implementation. In the near future, however, standard Pascal I/O will be implemented.

The reliability of the compiler is very good. This has been verified by the library of programs that are being written in Pascal here. We are making the effort to write new software in Pascal as its advantages over assembly language are obvious.

In pass 1 of our 7 pass compiler, we have implemented an automatic formatting option. This feature, implemented with very little compile-time overhead, rearranges the indentation of appropriate Pascal constructs in order to make the logical meaning of the program more evident. We have found this to be very helpful in communicating programs between different programmers as indenting style is preserved across programs. By incorporating this into the compiler these conventions are enforced.

Computer Automation would like to see the user's group strengthened so that standards are encouraged for program portability. This would facilitate the creation of a clearing house for Pascal Software tools as advocated by Mike Ball in #6. Areas such as I/O and compiler control options need to be standardized. I am interested in participating in the user's group and am willing to contribute to this effort as a representative of Computer Automation.

Sincerely,

Robert C. Hutchins

Computer Automation, Inc.
NAKED MINI® Division
18651 Von Karman
Irvine, California 92713
Telephone: 714 833 8830
TWX: 910 595 1767

CONTROL DATA CYBER 18

Jim Fontana (CDC) describes the CYBER 18 as a self-contained interactive system, and the compiler as being derived from the compiler for the CDC 2550 front end processor. Dennis Nicolai (CDC, Minneapolis) told us that the CYBER 18 and the 2550 have equivalent instruction sets, and that the compiler is a cross-compiler which runs on CYBER 70's and 170's. Code is linked and "down loaded" to the CYBER 18.

CONTROL DATA 6000, 7000, CYBER 70, CYBER 170

1. On January 31, 1977, Niklaus Wirth and Urs Ammann of ETH, Zurich, entered into a seven point agreement with Andy Mickel and John Strait at the University of Minnesota for the purpose of future maintenance of Pascal 6000. Maintenance duties will now be handled by Minnesota. We will continue to collaborate with Urs, Niklaus, Chris Jacobi, and Svend Knudsen, and other Pascal 6000 users in development of the Pascal 6000 system.

2. IMPORTANT: We are now soliciting local modifications and additions to the library that have been made to Pascal 6000 at your site (we are working on a Release 3). Please send a listing only to John P. Strait, UCC: 227 Exp Engr, University of Minnesota, Minneapolis, MN 55455, USA.

3. We would like to thank Wilhelm Burger (U. of Texas), Dave Tarabar (U. of Massachusetts), Gideon Yuval (Hebrew U.), Tony Addyman and Peter Hayes (U. of Manchester), Helmut Golde (U. of Washington), Richard Cichelli (Lehigh U.), Gary Carter and Ron Sheen (U. of Nevada), Tony Gerber and Carroll Morgan (U. of Sydney), and Michael Hagerty (ABT Associates) for already sending in listings.

4. As announced in Pascal Newsletter #5 we are still accepting bug reports.

5. We are soliciting listings of software tools.

6. Release 3 work is underway. Release 3 will appear no sooner than early 1978. These features are projected: improved compiler and run-time system, enhanced library, enhanced tools, documentation, and installation procedures.

7. Peter Hayes in a letter to Urs Ammann dated Jan. 18, 1977, suggested that the University of Manchester's 7600 mods to Pascal 6000 (derived from CERN's 7600 mods announced in #5) be included on the distribution tape. We intend to accommodate this in Release 3.

RECAU Pascal Manual by Jorgen Staunstrup and Ewald Skov Jensen, Regional EDP Center at the University Aarhus, Denmark (March, 1977, 177 pages), describes the CDC 6000 Pascal implementation with local extensions. Because Pascal is the most-used language there (!) a definitive description (better than Jensen and Wirth) was deemed necessary.

-Andy and John.

DATA GENERAL NOVA

University of Lancaster

Dear Andy

The Department of Computer Studies at Lancaster University has developed facilities for running PASCAL programs under the RDOS operating system on the Data General Nova series of computers. We are prepared to release these facilities as from 1st May, 1977, without any formal commitment to provide support.

Programs are compiled using the PASCAL-P4 compiler, which produces PCODE. This is then converted to binary form by an assembler (written in PASCAL), ready to be executed by an interpreter (written in NOVA assembly language).

Typical runtimes compare favourably with those of other languages generally available on the Nova.

Enquiries are welcomed from interested users: please contact Mr. R. E. Berry at the above address.

Yours faithfully,

R. E. Berry

Department of Computer Studies
Bailrigg, Lancaster
Telephone Lancaster 65201 (STD 0524)

DIGITAL EQUIPMENT PDP-10, DECSYSTEM-10, DECSYSTEM-20

Charles Hedrick (University of Illinois) and Wally Wedel (University of Texas) independently report that the improved PDP-10 compiler announced by Nagel in Pascal Newsletter #6 will be distributed by DECUS (Digital Equipment Computer Users Society) in Marlboro, Massachusetts.

DIGITAL EQUIPMENT PDP-11

We have been hoping to hear from Stephen Schwarm (coordinator of DECUS SIG Pascal) regarding the progress of that group. We have not yet received the group's newsletter. In view of the large number of implementations for the PDP-11, it appears that coordination is desperately needed. Anyone interested is encouraged to contact Schwarm at E.I. DuPont de Nemours Co., 101 Beech St., Wilmington, DE 19893 (302/774-1609).

Kenneth Bowles has announced a Pascal based stand-alone software system (including compiler, interpreter, editor, and interactive monitor) for the PDP-11/10 and the LSI-10 (also the Zilog Z-80). The system will be available in mid-summer through the UCSD Computer Center. The price has been set at $200 each. Compilation speed is 1000 lines/minute on the PDP-11/10 and 700 lines/minute on the LSI-11. The language processed is Pascal P2, extended with procedures for string processing and graphics applications. It processes the full ASCII character set, and allows sets of char. All systems support graphics display, keyboard, and floppy disk. For more information see Ken's article in this issue, or contact Ken at Institute for Information Systems, University of California San Diego, La Jolla, CA 92093 (714/452-4526).

There may be hope for UNIX users! Ken Bowles (above) tells us that a compile and go Pascal implementation has been written by Ken Thompson of Bell Laboratories. Can anyone tell us more? Also we have heard that Pierre Verbaeten and K.V. Leuren have an implementation. Their address is Applied Mathematics and Programming Division, Celestijnenlaan 200 B, B-3030 Heverlee Belgium.

On August 24, 1976, Jeff Schriebman (485 Cory Hall, U. of Calif. Berkeley, 94720) wrote to George Richmond (* who forwarded the letter to us on Feb. 10, 1977 *) that he has a Pascal interpreter running under UNIX on a PDP-11/70. We have received no reply to a follow-up inquiry (* Feb. 24, 1977 *). Richard J. Cichelli reports that Charles J. Printer of the University of California has a Pascal interpreter under UNIX, which has very tight code. PLEASE, can anyone help us track down these people or their implementations?

Wiley Greiner (TRW, Inc.), in a letter dated March 11, 1977, mentioned an implementation by Brian Lucas of the National Bureau of Standards which runs under DEC's RSX11d (V6.2) and RSX11M (V3.0). (* Come on Brian, don't be bashful. Please write to us. *) Wiley's address is Building 90-2178, TRW/DSSG, One Space Park, Redondo Beach, CA 90278.

PDP-11 PASCAL IMPLEMENTATION NOTE

Stockholm
1977-02-09

1 IMPLEMENTOR

Seved Torstendahl
Address:
Telefon AB LM Ericsson
AL/Ufe
S-125 26 Stockholm, Sweden

Phone number:
Sweden, 08 / 99 02 00 until 1977-03-31
08 / 719 00 00 from 1977-04-01

2 MACHINE

DEC-10: crosscompiler that generates code for all PDP-11's.
PDP-11: model 35 and up.

This version of the compiler does not generate code for floating point hardware or extended arithmetic. But the next version will do so when an option switch is set.

3 OPERATING SYSTEM

RSX-11M. (DEC-10 crosscompiler under TOPS-10). Probably it is an easy task to replace the RSX interfacing routines with new ones interfacing DOS or RT-11. We do not plan to do that work here. Maybe routines to interface with RSX-11S will be made.

4,5,6 DISTRIBUTION,DOCUMENTATION,MAINTENANCE

Not yet clear, but hopefully more information will be available soon. A user manual, complementing the Report, is under development.

RESTRICTIONS AND EXTENSIONS

The compiler is a modification of the crosscompiler from Mr Bron of Twente University of Technology, The Netherlands. Two major modifications have been undertaken:
- the compiler generates standard object modules
- the compiler gives full access to RSX file system

The following list is mainly a copy from Mr Bron's contribution in Pascal Newsletter #7.

With regard to the definition of Pascal in Pascal User Manual and Report the following restrictions hold:

- packed data structures are only implemented for character arrays (always packed, two char's/word) and for boolean arrays (packing optional, one boolean/bit). The procedures pack and unpack are not implemented.
- only local jumps are allowed.
- a pair of procedures, mark and release, to allocate and deallocate dynamic storage.

The following extensions have been implemented:

- function results can be of nonscalar type,
- arrays with unspecified bounds (but specified index-structure) can be used as formal parameters to procedures, allowing differently declared variables or constants as actual parameters,
- a string parameter type has been intoduced in which one-dimensional character arrays or substrings thereof may be passed as parameters. Such strings and their constituent characters are considered as "read only",
- procedures may be compiled separately,
- separately compiled procedures can be accessed through a declaration with the procedure block replaced by "extern".

SOURCE LANGUAGE

The compilers are written in Pascal, and both have the same source code except for two separately compiled routines. The crosscompiler is generated when the DEC-10 Pasrel compiler from Hamburg compiles the source. When it then compiles itself the PDP-11 version is created.

The size of the compiler is 50Kwords of code. In a PDP-11 running under RSX-11M V2 only 32 Kwords are available for code and data. Through a slight modification of the overlay loading routine of RSX-11M it has been possible to segment the very recursive compiler. It now fits in a 32 Kwords partition and uses about 22 Kwords for code leaving 10 kwords for data.

RELIABILITY

Good. The reliability of the original crosscompiler was very good.

10 METHOD OF DEVELOPMENT

The crosscompiler for PDP-11 running on DEC-10 produced by Bron et al was used as input. As mentioned earlier, this compiler was modified to generate object code linkable under RSX-11M and to give access to the file system of RSX-11M. When the crosscompiler was finished it compiled itself and the compiler was thus transferred to PDP-11.

The implementation effort until now is about 5 manmonths. To make use of floating point hardware another two manmonths will be necessary. Probably a new version which performs some optimization will be developped later.

Dear sir:

At my installation, we are presently using the ElectroScientific Industries implementation of PASCAL on three different PDP-11 processors, all using the RT-11 operating system. These machines are a 16K 11/05, a 28K 11/10, and a 28K 11/40 with FIS. Our applications are in speech recognition, real-time simulation, and computer graphics using Evans & Suntherlund Picture Systems. We have found the ESI PASCAL to be much faster than DEC FORTRAN, and very economical in core requirements. Our worst case benchmark involved a "number-crunching" program translated almost literally from FORTRAN-for this benchmark, the ESI PASCAL executed about 40% faster than FORTRAN, while requiring about one third the core for execution. Much of the core improvement is due to the small support package required for ESI PASCAL, as opposed to the somewhat larger requirements of DEC FORTRAN.

We have found that we can compile quite large programs even on our 16K 11/05. We have compiled 3000 line programs in 28K on the 11/40. At my request, since our applications involve graphics using programs with in MACRO to expect FORTRAN calling sequences, ESI have added the capability to declare procedures "external FORTRAN"'. We have successfully used this feature to communicate with the Evans & Suthland graphics software in a production environment. ESI also offers an optional optimizer, and a formatting/cross reference package.

Reliability of the compiler has been far better than the DEC FORTRAN system which has been completely replaced at our installation. The vendor seems to be responsive in terms of support.

We have ordered PASCAL for our XDS Sigma 9 and DEC 11/70 running UNIX. I will inform you of our success with these implementations.

I am interested in implementations of PASCAL on DG NOVA, HP 21MX, DEC PDP-12, SEL840MP.

DAVE KAISER.

Zay B. Curtis
Project Manager

zbc

P.O. Box 235
Moffett Field, CA 94035

University of Illinois at Urbana-Champaign

DEPARTMENT OF COMPUTER SCIENCE
Urbana, Illinois 61801
(217) 333-4428

January 21, 1977

Mr. Timothy Bonham
Pascal Implementations
University Computer Center
227 Experimental Engineering Building
University of Minnesota
Minneapolis, MN 55455

Dear Mr. Bonham:

This letter is in response to your October 25 inquiry concerning the University of Illinois Pascal effort.

Our normal procedure upon receipt of a specific inquiry for Pascal-11 is to continue correspondence via a standard letter (see enclosure A). That letter provides a rough description of our compiler and details the method by which a "legitimate" party may obtain a copy. In particular, it is required that the recipient agree to the conditions set forth in Professor Snyder's letter (see enclosure B). The question of whether any specific usage is deemed "research, education or other legitimate purpose" or whether it is deemed "commercial" is one that can be answered only by the University of Illinois Administration and/or the National Science Foundation.

The Pascal-11 compiler was developed by A. Ian Stocks and Jayant Krishnaswamy* under the direction of the late Professor Donald B. Gillies. It was originally intended solely for in-house use as both a systems programming language and a pedagogical tool. However, increased outside interest resulted in fairly widespread distribution of various versions of the compiler. Consequently, we have found it necessary to freeze the distribution version as described in enclosure A. In particular, our distribution version does not implement WITH, variant records, arrays of records, procedures-as-parameters, and type SET.

In addition, we have made a number of extensions to the language which we have found to be quite useful. Most of these extensions are mentioned in [1]. Unfortunately, there is no documentation beyond that for these extensions.

Since the project under which the compiler was developed has expired, we have no source of funds for maintaining and upgrading the compiler. Consequently, we offer Pascal-11 "as-is," with no plans** to extend it or to implement it on other systems. (However, we have word that others (besides ESI) have transported Pascal-11 to RT-11 and DOS/V9.) Finally, we have no magnetic tape or RK style disk facilities, and the clerical personnel who perform the distribution service are trained to simply copy from one DECtape to another. Therefore, PDP-11 formated DECtape is our only mode of distribution.

Sincerely yours,

M. Dennis Mickunas
Assistant Professor of
Computer Science

MDM:clg

Enclosures

*Present addresses: Professor A. I. Stocks, Department of Computer Science, University of Southwestern Louisiana, P. O. Box 4330, Lafayette, Louisiana 70509; J. Krishnaswamy, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801

**Read: "plans which will result in and updated distribution version." We, of course, are continuing with unsponsored research, which includes: 1) implementing full Pascal; 2) Professor Roy Campbell's extending Pascal-11 to include "Path Expressions" (Pathcal??), and; 3) transporting Pascal-11 to modified RT-11 and MERT (mini UNIX).

1. Stocks, A. I. and J. Krishnaswamy. "On a Transportable High Level Language for Minicomputers," ACM SIGMINI-SIGPLAN Interface Meeting: Programming Systems in the Small Processor Environment, New Orleans, March 1976.

ENCLOSURE A

This letter is in response to your inquiry concerning our PDP-11 PASCAL compiler.

Our PASCAL-11 compiler and the associated package of run-time routines operate under our own operating system, which grew out of DEC's DOS/V4. While our PASCAL-11 system is not yet complete enough for widespread distribution, we are happy to make it available on a limited basis to interested persons. Our distribution package includes:

1) PASCAL-11 source of the PASCAL-11 compiler;
2) MACRO-11 source of the PASCAL-11 run-time routines;
3) binary for both the compiler and the run-time routines, and;
4) binary for our operating system.

In case you desire to install PASCAL-11 on your own version of DOS, we also provide a list of DOS/V4 modifications. We believe that these modifications are sufficient for adapting DOS/V4 to PASCAL-11, but we can, of course, make no guarantees. We caution that these modifications are not sufficient for installing PASCAL-11 on other operating systems, but your DOS expert should be able to make the necessary modifications using our DOS/V4 modifications as guidelines.

Hardware requirements for executing the compiler are: a PDP-11/20 (or higher) processor with 28K words of addressable core storage, and either 1) a DEC RF-11, or; 2) a DEC RK-11. In case you have some other disk, your DOS expert should have little trouble replacing our disk driver with your own. In addition, it is necessary that your system be able to read DECtapes, since that is our only mode of distribution for the PASCAL-11 system.

The present version of our PASCAL-11 compiler does not implement WITH or type REAL or SET, nor does it permit variant records or procedures-as-parameters. Our version is otherwise essentially in accord with the Revised Report, except that we have preserved EOL in lieu of WRITELN/READLN, and we have incorporated some extensions, including compile-time options, source level library routines, and overlays. Documentation for the compiler is, unfortunately, very sparse at present, but we shall include in the distribution package all that is available.

The PASCAL-11 compiler was developed at the University of Illinois at Urbana-Champaign and is copyrighted by its Board of Trustees. This work was supported, in part, by NSF Grant DCR 72-03740 A01 to the University of Illinois at Urbana-Champaign. Accordingly, distribution is made to any interested persons or

parties who intend to use this software for "research, education, or other legitimate purposes." The NSF requires that we inform them of those receiving this software and their intended uses of it. Consequently, if you are interested in obtaining this software, please mail

1) three (3) DECtapes (These must be in PDP-11 format!);
2) a statement of your intended uses;
3) one signed copy of Professor Snyder's enclosed letter, and;
4) a stamped, self-addressed mailer for returning your DECtapes (total weight is about 2 pounds)

to

PASCAL-11
c/o M. D. Mickunas
222 Digital Computer Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Upon receipt of the above items, we shall return your DECtapes with a copy of our distribution package.

HEWLETT PACKARD HP-2100

Mattia Hmeljak (U. of Trieste) wrote (* Feb. 5, 1977 *) to say that his group intends to implement a P-Code interpreter in HP-Algol for the HP-2100. He asks that anyone else working on an implementation for this machine contact him at Istituto di Elettrotecnica ed Elettronica, Universita di Trieste, Trieste-Italy.

HONEYWELL SERIES 66

Janis Zeltins of Honeywell Information Systems, 7400 Metro Boulevard, Edina, MN 55435 (MS-1104), informed us of the availability of documentation for the Honeywell implementation running under the GCOS operating system. "A Pascal Product Brief" (publication #AW66, free) is a 1 to 2 page marketing oriented piece. A fuller description of the implementation is "Pascal User's Guide" (manual #AW65, $1.30; about 30 pages). These are available through Honeywell Information Systems, Attn: Publication Services MS-339, 40 Guest Street, Brighton, MA 02135.

IBM 360, 370

Obviously there is a crisis with IBM implementations (just as bad or worse than PDP-11's!). After having evaluated both the SUNY Stony Brook and the University of Manitoba compilers (the two most widely distributed), and found them to be disappointing, two new projects were begun. Albrecht Biedl at the Technical University of Berlin and a group at Imperial College, London, independently embarked on new implementations based on Pascal P4. Dissatisfaction with the Stony Brook and Manitoba compilers exists elsewhere, although a good report on Stony Brook's comes from David Gomberg at American University, Washington D.C., and an expression of contentment with Manitoba's comes via the University Computer Center Newsletter, U. of Southern California, Los Angeles.

David Gomberg reported to us on Feb. 22, 1977, that "anyone waiting for a clean supported version of the HITAC 8800 Pascal for use under OS/360" can stop holding their breath. He received a letter from Teruo Hikita to the effect that they feel some unwillingness to distribute the system formally because of lack of full support, I/O routines coded in Fortran, lack of IBM compatible load modules, and inability to support the system "formally and continuously."

However, Joseph Mezzaroba, Villanova University (215/527-2100, x669), reported on April 16, 1977, that he recently coaxed a copy of the HITAC compiler from Hikita and had it running for 3 weeks under DOS. The I/O routines are being rewritten in assembler. Joseph is very happy with the system - it produces good code and has been extremely reliable in its initial use by 60 graduate students (1500 jobs). HITAC Pascal compiles twice as fast as PL/I and executes 5 times faster than PL/I under DOS according to Mezzaroba.

Currently we still have the Tokyo (HITAC 8800) compiler, SUNY Stony Brook, U. of Manitoba (no news for 8 months!), New Mexico Tech., Stanford Linear Accelerator Center, U. of Grenoble, Oslo Heart Hospital, and now the U. of British Columbia implementation. Add the other two European efforts and we have ten major implementations on the IBM 370!

-Andy Mickel

Dear Andy:

This note will serve to describe our Pascal implementation for the IBM 370/168 running under the MTS operating system. I've also enclosed a copy of our User's Guide.

1. Names, etc., of implementors, maintainers, etc.

   Professor Bary W. Pollack
   Professor Robert A. Fraley
   Department of Computer Science
   University of British Columbia
   Vancouver, British Columbia
   Canada V6T 1W5
   604-228-6794, 604-228-3061

2. Machine, manufacturer, etc.

   IBM 370/168. The machine operates in university environment with heavy background and moderate interactive loads. The translator should be compatible with most large IBM 360 or 370 series machines. Current development uses the MTS operating system.

3. Operating system, minimal hardware configuration.

   Operates under the Michigan Terminal System, MTS. The monitor may be modified with minimal effort to run under VS, OS, etc. The translator requires about 320,000 bytes of core. Standard OS object modules are generated. An obsolete OS monitor is available; we hope to have it updated to work with the current compiler shortly.

   Division of the compiler into overlays for non-VM systems would be possible.

4. Method of distribution.

   The current version is available for distribution now. Distribution will be via 9-track magnetic tape. Costs will be limited to postage (and tape purchase, if one is not supplied).

5. Documentation.

   Documentation consists of a User's Guide containing a complete description of the language's departures from the Jensen and Wirth Pascal User Manual and Report.

6. Maintenance policy.

A maintenance policy has not yet been decided upon. It is anticipated that periodic upgrades and modifications will be distributed at least once a year. Reported bugs will be corrected as quickly as possible with notification to users.

7. Standard Pascal?.

The compiler provides numerous extensions and a few restrictions. A compiler option issues error messages when non-standard features are used. A complete description is contained within the documentation provided. A summary of the differences follows.

Extensions:

Strings are padded on the right with blanks.
There is a "CASE" default label: "<>"
Optional ";" allowed before "ELSE".
"(...)" may be used instead of "[...]".
The EOL character has been retained.
"PACKED" is ignored.
Additional built-in functions:
MIN, MAX, SUBSTR (using constant length), POSITION (provides direct-access I/O), I/O interface functions and extensions to RESET and REWRITE, INSERT function for data-packing.
Input of character strings using READ.
Support of EBCDIC characters ¬, &, and |.
Use "..." for comments.
"VALUE" section exists for variable initialization.
Hexadecimal integers supported.
FORTRAN subroutines may be called. A return code is available in the standard variable RCODE.
Direct access files.

Restrictions:

Sets currently limited to 0..31.
PROGRAM statement not used.
FILEs may not be components of other structures.
<Expression>..<expression> is not allowed in sets.
INPUT@ is initially EOL instead of the first character of the file. This is transparent when READ is used.
DISPOSE is not implemented.

Projected extensions:

McCarthy IF.
OR and AND lower precedence than relations;
"usual" precedence used throughout.
Sets over range 0..255.
Better control of input anb output formats.

8. Compiler, size, implementation, speed, etc.

The translator is written in Pascal and is modeled after the CDC 6400 implementation, but it has been extensively modified and improved. The translator consists of approximately 8000 lines of Pascal code. The run-time library consists of approximately 500 lines of Pascal code. The monitor (which contains the interface to the operating system) consists of approximately 2000 lines of IBM Assembler G code. The translation speed has not been determined, but it seems faster than our Algol-W compiler. The code produced has been timed against Algol-W code and is almost uniformly 10-15% better. This is especially true of any program using a large number of procedure calls. The compiler compiles itself in less than 60 seconds of 370/168 processor time.

9. Reliability, etc.

The reliability to date has been excellent. A student version of the translator has been running since September, 1976, with only one detected compiler error. The main system version has been in operation since December, 1975. All problems which have been encountered to date have been corrected.

10. Method of development.

The original translator was developed by Wirth and several graduate students at Stanford University as a partial re-write of the CDC 6400 version in 1972. The current translator and monitor have been extensively modified, a run-time library has been implemented, and a post-mortem symbolic dump package has been developed. The translator has been under continuous development at UBC since December, 1975, by two faculty members and one graduate student.

Best regards,

Feb. 4, 1977

Barry W. Pollack & Robert A. Fraley
UNIVERSITY OF BRITISH COLUMBIA
Department of Computer Science
2075 Wesbrook Place
Vancouver, British Columbia
Canada V6T 1W5

Pascal/360 now available to DOS users

The Stony Brook Pascal/360 compiler (announced in SIGPLAN Notices, Feb. 1976) has been given a DOS interface. It has been installed and tested on an IBM 370/135 running DOS/VS release 32. This version of the compiler is identical to Pascal/360 Release 1, Update 3 (OS) except for the operating system interface. At present, the main storage requirement is a 150K partition. A small-partition edition for DOS/360 users is planned for summer, 1977 release.

The distribution tape, installation instructions, and copies of all future maintenance updates and documentation are available for a one-time fee of $175. A User's Guide is also available in quantities at $1 per copy. For complete information, please write to:

Pascal Compiler Project
Dept. of Computer Science
State University of New York
at Stony Brook
Stony Brook, New York 11794

IBM 1130

O. Lecarme wishes to apologise for an error in issue #6 regarding the IBM 1130. He incorrectly informed us that an implementation was completed at the University of Neuchatel. Instead, he says, a Pascal-S compiler (not interpreter) has been implemented for the 1130 by Helmut Sandmayr, Nev Technikum, CH-9470 Buchs, Switzerland.

ICL 1900 SERIES

The current ICL 1900 series compiler was developed over the period 1974-76 by Jim Welsh, Colum Quinn, and Kathleen McShane, at the Department of Computer Science, Queen's University, Belfast BT7 1NN Northern Ireland, U.K. This project was a complete rewrite of the old ICL 1900 compiler (famous for being the first to exist outside of Zurich). Improved code and internal design, implementation of the revised report, and improved diagnostic facilities were goals achieved by the new compiler, known as the MK2. The ICL MK2 compiler is distributed by Jim to nearly 50 sites, mostly within the U.K.

Its performance compares very favorably to Fortran on the ICL 1900. The compiler requires nearly 32K to run and has been installed under various operating systems: George 3, George 4, Executive, MAXIMOP, and COOP.

The most interesting feature of the MK2 compiler from an implementor's point of view is that it has been designed to be ported to other machines. Specifically, the semantic analysis and code generation parts have been cleanly separated. Thus it can be used as a bootstrap compiler for other machines and can be likened to the Pascal Trunk compiler. (See the ICL 2900 section below.)

Diagnostic enhancements to the MK2 were provided by David Watt and Bill Findlay, Computer Science Department, University of Glasgow, Glasgow G12 8QQ Scotland, U.K. Their diagnostics system includes a post-mortem dump with array, record, and file-status variables displayed; an execution profile shows the number of times each line of a program is executed, and retrospective and forward traces of the exact source statements executed.

Documentation (in the form of a supplement to the revised report) for the ICL 1900 implementation from Glasgow (dated Feb. 23, 1977, 27 pages - very clearly written) indicates that the ICL 1900 6-bit character set is used, sets may have 48 elements, files are not allowed as components of any structured type, and non-discriminated variant records were removed.

-Andy Mickel

ICL 2900

One project based on the ICL 1900 MK2 compiler (above) is to produce a compiler for the upper ICL 2900 series machines (2970, 2980) at the University of Southampton, supported by David Barron, Judy Mullins, John Goodson, Mike Rees, and Andy Schulkins. The compiler is largely being written by John Reynolds with the aid of Jules Zell, both of the Imperial College, Department of Computing and Control, London SW7, U.K. John Reynolds rewrote the code generators for the 2900 which is stack-oriented and possesses a completely different architecture than the 1900 series machines. Poor computer system performance of the 2970 at Southampton led to the decision by John to develop the compiler on the Control Data 7600 at the University of London Computing Centre, making use of Urs Ammann's Pascal 6000. After compiling Jim Welsh's compiler with Urs Ammann's compiler and fixing the (only!) ten errors which resulted (out of 9000 lines of code) John was able to generate assembler for the 2970 which he ported to Southampton and successfully loaded and ran test programs. John remarked that the event of sucessfully moving to the 7600 "said something about each of Urs Ammann's compiler, Jim Welsh's compiler, and Pascal the language."

To circumvent an unwieldy ICL operating system on the 2970, the University of Edinburgh scientific jobber will probably be used to harbor this Pascal compiler.

-Andy Mickel

NB: David Joslin of Sussex University Computer Centre, Falmer, Brighton, Sussex, U.K., is coordinating this consortium of universities by acting as a clearinghouse for all ICL compilers. Anyone having ICL news should forward it to David who is in close contact with the Pascal Newsletter.

INTEL 8080

Feb 22, 1977

Mr. Andy Mickel
Pascal Implementations
University Computer Center : 227 Exp Engr
University of Minnesota
Minneapolis, Minnesota 55455

Dear Andy,

This letter is in response to our recent telephone conversation regarding sequential Pascal for the Intel 8080A microcomputer.

The sequential Pascal compiler, written by Per Brinch Hansen and Alfred C. Hartmann of Caltech, generates code for a virtual machine. I have simulated the virtual machine with a real machine, the Intel Intellec Microcomputer Development System (MDS). My PAS80 program, which is the implementation of the virtual machine, is written in the high level language PL/M-80. Emulating a 16-bit virtual machine using PL/M-80 on the 8-bit Intel 8080A certainly did not produce a high-speed real machine. However, I feel that compilation and execution speeds are tolerable for the purposes of beginning work with Pascal on the 8080A.

At this time the 7 pass sequential Pascal compiler has been successfully self-compiled on the microcomputer (9,835 lines).

I will use the checklist provided in the Implementation Notes section of your newsletter to provide you with further information:

1. Implementor:

   Thomas A. Rolander
   1012 Smith Ave.
   Campbell, Ca. 95008
   (408) 378-5785

   Distributor:

   INSITE
   Intel User's Library
   Microcomputer Division
   3065 Bowers Ave.
   Santa Clara, Ca. 95051
   (408) 246-7501 x2948

2. Machine: Intel 8080A using the
   Intel Intellec Microcomputer Development System

3. Operating System: Intel MDS ISIS-II

   Hardware Configuration: 64K Bytes of RAM
   Dual Floppy Disks

4. Distribution Format:
   The software is distributed on two soft-sectored diskettes containing: the PAS80 program, the sequential Pascal compiler in virtual machine code form, the PL/M-80 source code for PAS80, and the source code for the entire 7 pass sequential Pascal compiler written in sequential Pascal.

5. Documentation:
PAS80 documentation is supplied in the form of a short User's Guide, syntax graphs, and the source code for the virtual machine implementation and the compiler.

6. Maintenance Policy:

NONE, however bug reports will be accepted.

Future Development Plans:
The initial version of PAS80 does not support floating point operations. However, all of the required hooks have been incorporated, facilitating the implementation which is currently in progress.
Work is also in progress to reduce memory requirements from 60K to 32K bytes.
Direct machine code generation for the Intel 8080A is being considered.
The possibility of a concurrent Pascal implementation is also under consideration.

7. Pascal Implementation:
Complete sequential Pascal, as described by Per Brinch Hansen, has been implemented with the exception of floating point operations.

8. Compiler Characteristics:

-Interpreter, written in PL/M-80,
1300 lines of source code,
10K bytes.

-Speed,
30 lines / minute.

9. Reliability of Compiler:
Unknown, however it will self-compile and has been used successfully by students, providing reasonable diagnostics and error recovery.

10. Method of Development:
The virtual machine implementation was coded in PL/M-80 and then debugged using the virtual machine code files of the sequential Pascal compiler itself to compile small test programs, and then finally the compiler was self-compiled.
The implementation required about 2 man-months-of-evenings and was accomplished in my spare time. It could have been completed in about 2 1/2 weeks on a full-time basis.
I was familiar with the process of implementing the virtual machine from previous experiences on the PDP 11/40 under RSTS/E and with the TI 9900. Credit for the ease of implementation is due to Per Brinch Hansen who developed the virtual machine.

In summary, while compilation and execution speeds are slow, this implementation does provide a tool which can be used for further Pascal developments on microcomputers.

Thomas A. Rolander

Thomas A. Rolander

UNIVERSITY OF MINNESOTA
TWIN CITIES

Peter Zechmeister
Microcomputer User's Group (UMMUG)
Department of Electrical Engineering
139 Electrical Engineering
123 Church Street S.E.
Minneapolis, Minnesota 55455

IMPLEMENTOR

Peter Zechmeister
Microcomputer User's Group (UMMUG)
Dept. of Electrical Engineering
123 Church Street S. E.
Minneapolis, Minnesota 55455
I am responding to the request of new implementors. If anyone needs more specific information please write me.

MACHINE - Intel 8080 8-bit microprocessor

The target machine for this implementation is an Intel 8-bit microprocessor. With easy modifications it can be adapted to run on most microprocessors.

OPERATING SYSTEM AND HARDWARE CONFIGURATION - Target Machine

The compiler includes a high level operating system which interfaces between the user, software, and hardware in a simple but powerful syntax. The minimum configuration consists of a console I/O device (TTY), about 16K memory for the compiler to reside in (Not yet completed. If the cross compiler is used only the OS is needed; 2K memory + users program.). Note that the compiler, user programs, and OS may reside in ROM, since code is seperate from the variable space.

METHOD OF DISTRIBUTION

None at this time but possibly late this summer.

DOCUMENTATION AVAILABLE

Being worked on.

MAINTENANCE POLICY

Being worked on.

STANDARD PASCAL

This implementation, which I call Tiny Pascal (TP), may seem a little barbaric compared to Pascal 6000, but this compiler was written with the microcomputer in mind and is an improvement in software for the small computer user. I would also like to add that this compiler (system) represents the minimal language and is meant to be a systems implementation language as well as a low level programming language which can be expanded with minimal effort. The complex data structures and variable types were left out of the compiler in order to fit the compiler on a micro in a reasonably small memory. The data types may be added in a future Extended Tiny Pascal (ETP). Also there exists the bootstrap compiler which is being used to generate TP and the monitor routines (all written in Tiny Pascal), and is written in Pascal 6000 which produces 8080 code.

IMPLEMENTATION

This is a true compiler that produces 8080 code. The Pascal 6000 bootstrap compiler is around 2500 lines long and loads in about 14K. The TP compiler is around 1500 lines long and loads in about 14K (Not yet finished.). The compiler route was taken because an interpreter system is too slow for most real-time lab situations even though they are smaller. This is also an excellent language for hardware design by manufacturers, allowing bit fiddling but yet still a high level language in a reasonable amount of memory. (* The cross-compiler runs at 2400 lines/minute on a CDC 6400. *)

RELIABILITY

The reliability of the compiler is excellent, an efficient register mapping algorithm is incorporated into the compiler.

METHOD OF DEVELOPEMENT

The original compiler was developed from PLO (Taken from the book Algorithms + Data Structures = Programs by Niklaus Wirth.). A considerable amount of modifications was done to implement variable types, Pascal statements, code generation, and register mapping.

The TP compiler (bootstrap) currently produces good runnable code but documentation and a few loose ends remain to be taken care of. I am currently considering the writting of Modula for 8080 based microcomputers, since TP could be used as a starting point.

Sincerely,

Peter Zechmeister

Peter Zechmeister

INTERDATA 4

Jean Vaucher of the University of Montreal has informed us in a letter dated Dec. 13, 1976, that the Interdata 4 project there has been discontinued because of the availability of Pascal on other machines.

MOTOROLA 6800

Mark Rustad has provided us with some changes (received April 4, 1977) to his notice which appeared in Pascal Newsletter #6. Under Checklist point 7, he indicates that the following features have been added or restored: case statement, variant records, enumeration types, for statement, the type real (as a four byte quantity), and an exit statement (which returns from a procedure or function). Mark lists the deviations from standard Pascal as being:

1. No declared files; get, put, reset, and rewrite are not supported.
2. The with and goto statements are not supported.
3. The standard procedures sin, cos, arctan, exp, ln, sqrt, pack, and unpack are not supported.
4. The case statement has an optional else clause.
5. The predefined procedure exit is non-standard.

Mark also says that the compiler code occupies about 19K-20K bytes, while his M-CODE interpreter takes about 3K (including a floating point package). He is currently working on optimization features for the compiler.

NANODATA QM-1

TRW

Ref: 6201.DMH-016

17 March 1977

Implementor: Dennis Heimbigner
TRW DSSG
Mail Station: R3/1072
1 Space Park
Redondo Beach, CA 90278
(213) 536-2914 or (213) 535-0833

Machine: Nanodata QM-1 with (minimum)
256 words nanostore
8K words control store
60K words main store
9755 55 megabyte disk
TASK version 1.04.02 or later
PROD version 2.04.01 or later

Optional:

Card reader
Printer (highly desirable)

Documentation: a. Brinch Hansen's SOLO manuals (not available thru TRW)
b. Short machine readable document describing the implementation and ways to modify it.

Reliability: In-house use has been light but the system has been good. to the extent we have used it.

Method of Development: The Concurrent Pascal system kernel was programmed in micro-code. Some care was taken to insure that the QM-1's virtual machine was compatible with the virtual machine defined by the PDP-11/45 kernel. Please note that I did not implement a PDP-11/45 emulator. As a result, virtual code object files (e.g., type SEQCODE or CONCODE) which run correctly under the PDP-11/45 system should run under the QM-1 system. The reverse is also true for programs which do not use the fact that integers on the QM-1 are 18 bits as opposed to 16 on the PDP-11.

The kernel was micro-coded in about 6 months, from January 1976 to June 1976, on a part-time basis. Some one half of that time was spent on the IO drivers.

Speed: Appears to run at about one-third the speed of the PDP-11/45 system. I believe that a modest programming effort could achieve parity in speed.

Distribution: Release by TRW is currently under consideration. Inquiries are welcome.

Sincerely,

Dennis Heimbigner

Dennis M. Heimbigner

DEFENSE AND SPACE SYSTEMS GROUP OF TRW INC. • ONE SPACE PARK, REDONDO BEACH, CALIFORNIA 90278 • (213) 535-4321

NORSK DATA NORD-10

A first version of PASCAL is now running on the NORD-10 under the MOSS operating system. This note gives a short introduction to the PASCAL system and how to use it.

NORD-10 PASCAL

The compiler has been developed from the P-PASCAL compiler by the following group:

Andora Fjeldsgaard
Petter Gjerull
Stein Gjessing
Jan Husemoen
Ketil Moen
Terje Noodt

The implementation is described in "Rapport om implementering av PASCAL på NORD-10", University of Oslo, April 1976.

The compiler utilizes the 2-bank feature of the NORD-10, so it is possible to run 64K programs with 64K of data. The present version compiles to symbolic assembly code, so that a compiled program must be assembled by AMORAL before it can be executed.

Non-implemented features

Compared to the full PASCAL language, the following are the main restrictions in NORD-10 PASCAL:

1. packed is not implemented (the compiler does however accept the symbol PACKED).
2. The type file is not implemented.
3. Formal procedures are not implemented.
4. Range and index checking are not implemented.
5. Arithmetic overflow is not checked.

How to use the system

The compiler is activated by the command

```
)*PASCAL
```

After the compiler has been loaded it will ask the user to specify which logical units are to be used for input, listing and compiled code. This conversation takes the following form:

```
INPUT =
  <specify octal unit number of source code file>
OUPUT =
  <specify octal unit number of listing file>
PRR =
  <specify octal unit number of the file where compiled
  code will be written>
```

The files should be opened before activating the compiler, but it is also possible to exit from the compiler by CTRL A, open the file, and then continue with the)GO command.

When compilation is finished (signalled by right parenthesis), the file containing the compiled code can for instance be saved for later use. To execute the program, go through the following steps:

1 Open the compiled code file with logical unit 3 (if not already open on this lun).
2)*LINKP
3)GO

NB: A PASCAL program will store some of its data at high adresses. Thus a text input for editing will not be preserved through a PASCAL compilation or execution of a PASCAL program.

The compiler recognizes the following options (placed within a comment and preceded by $):

C	Produce code - default is off
T	Produce tables of variables - default is off
L	Produce listing - default is on

In a PASCAL program the programmer can use the following file names:

```
INPUT      (default input file)
OUTPUT     (default output file)
PRR
PRD
```

The files that are used should appear in the program heading, as f. ex.:

```
PROGRAM PROG(INPUT,PRR);
```

Before data access to a file the program must call
RESET(<file name>)
for an input file, and
REWRITE(<file name>)
for an output file. These calls have the effect of writing the filename followed by an equal sign to the terminal, whereafter the logical unit number (octal) of the file can be specified.

For the files INPUT and OUTPUT the calls to RESET and REWRITE are done automatically if they appear in the program heading.

Machine dependant characteristics

1. A set can have up to 64 elements.
2. A procedure cannot have more than 253 words of local variables, including parameters, but excluding record and array variables.
3. An integer variable occupies 1 16-bit word, a floating variable 3 16-bit words.
4. A string can have a maximum length of 16 characters.

Improvements and changes

It is expected that the PASCAL system will be improved and changed frequently in the near future. A description of any change or improvement will be written on the file *PASCINF, which may be inspected or listed by the PASCAL user.

Questions, comments and error reports are invited, and can be given to any member of the PASCAL group.

Terje's address is Computing Center, University of Oslo, Blindern, Oslo 3, Norway.

ORGANISATION EUROPÉENNE POUR LA RECHERCHE NUCLÉAIRE
CERN EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

SIÈGE: GENÈVE/SUISSE

CERN LABORATOIRE I

Adresse postale / Postal address:
1211 GENÈVE 23
SUISSE / SWITZERLAND

Votre référence
Your reference

Notre référence
Our reference PS/CCI/DB/afcs

Andy Mickel
PASCAL Users Group
UCC: 227 Exp. Engr.
University of Minnesota
Minneapolis, MN 5545
U. S. A.

Geneva, 19th January 1977

PASCAL NEWSLETTER

Dear Andy,

I am pleased to announce the successful implementation of a Standard PASCAL compiler on the Norsk Data NORD-10 computer (running under SINTRAN III o/s), by myself and my colleague Robert Cailliau. We developed our compiler from the Zürich P4 code compiler, first assembling the source P4 code into relocatable binary P4 code and then interpreting it as efficiently as possible by an assembly code program. It is a great tribute to Professor Wirth and his team at Zürich, who have produced a most exceptionally concise description of the implementation procedure and a very readable compiler written in PASCAL, that we were able to implement our system in about 2 man months. Apart from a small problem with the character set (why do CDC have to be different to everybody else?), the implementation went like a dream.

The very professional polish to the compiler and its documentation, plus experience with its use, indicate that the compiler itself is extremely reliable, and since our assembler/interpreter is very simple in terms of coding and has successfully compiled the compiler, we have considerable faith in our system. Naturally, it is not ultra-fast, but nevertheless takes only 15 minutes to compile the compiler, which for a 16-bit minicomputer with 2 µsec cycle time is not too bad.

Naturally, anyone is welcome to receive a copy of our system, although the NORD-10 is currently used exclusively in Europe.

A quick word on the PASCAL language itself - I feel that when Professor Wirth stopped just short of creating the long sought after "obvious" replacement to FORTRAN as the standard language, he missed a great opportunity. Naturally unable to be the perfect all-time language, it does have some slight drawbacks (frequently discussed in this newsletter, and in particular no interface to external routines), most of which it would seem could be relatively easily overcome, but which, however, do make it more difficult than it should be to persuade users to take it up.

Anyone interested in our PASCAL system can contact :

David Bates
PS/CCI Group
CERN, 1211 Geneva 23
Switzerland (tél. 41-98-11)

Sincerely,

David L. Bates

David L. Bates

ENSMIM

ÉCOLE NATIONALE SUPÉRIEURE DE LA MÉTALLURGIE
ET DE L'INDUSTRIE DES MINES DE NANCY

ÉCOLE DES MINES, Parc de Saurupt 54042 NANCY CEDEX TÉLÉPHONE (28) 51.42.32
TELEX : ENSMIM 850 661

NANCY, le February 2, 1977

P U G
c/o Timothy Bonham
University Computer Center
227 Experimental Engineering Building
Minneapolis, Minnesota 55455
USA

Dear Tim:

As announced to George Richmond we are (still) working on implementing a Pascal compiler for the SEMS minicomputer series. To answer your 10 questions see attached implementation notice.

Our hope is to provide an entire implementation with efficient debugging tools for the programmer on a small computer. As all the available documentation on this project is writen in french, we think it better to send it directly only to people who ask for it, and I enclose one copy of it for your own use.

Yours sincerely,

A. Tisserant

SEMS T1600 / SOLAR PASCAL IMPLEMENTATION

1: Implementor : Alain Tisserant
Département Informatique de l'INPL
Ecole des Mines
Parc de Saurupt
54042 Nancy Cedex FRANCE
Tel.: (28) 51 42 32

2: Machines : SEMS T1600 and SOLAR 16/05/40/65

3: OS : BOS-D
Hardware required : MTS16
FHE or MHU disk
16 K words of core memory (minimum)

4,5,6 : Compiler not yet available. Will be distributed by IRIA.

7: Fully implements standard Pascal; also compatible with the IRIS 80 Pascal compiler.
Its extensions are character strings
LOOP ... EXIT ... END statement
I/O for sets and scalars symbolics
It allows also separate compilations, insertion of ASM or Fortran routines, and sets of any interval of integers.

8: Pascal is compiled in two passes, with intermediate language use. Of course, compilers are written in Pascal; the intermediate language is an adaptation of P-code for minicomputers. This implementation provides a fully transparent virtual memory.

9: Reliability: expected to be excellent!

10: P-code has been adapted for non-stack, 16 bits words, based addressing and accumulator machines. An automatic segmentation mechanism will allow compilation and execution of large programs (such as the compiler) with small memory requirements.
First pass of the compiler is parametrizable, but second pass must be hand rewritten for each implementation.

SIEMENS 4004, 7000 SERIES

SIEMENS PASCAL BS2000 PROGRAMMING SYSTEM.

A PASCAL Compiler for SIEMENS 4004/151 and all SIEMENS series 7000 installations running under operating system BS2000 has been developped by Dr. Manfred Sommer (Dept. D AP GE - SIEMENS AG - MUNICH - GERMANY) on the basis of the ETH P4 Compiler.
The Compiler may be used in an interactive Edit, Compile and Go environment, as the Compiler produces code that may run without relocation anywhere in virtual memory.
The interactive environment is provided by a PASCAL program 'dialogue' which invokes the Compiler and/or generated programs by an additional standard procedure :execp (i.e. invoke PASCAL program). This procedure may be used by all PASCAL programs and supplies the possibility of a nested execution hierarchy of PASCAL main programs.
The code produced by the Compiler (the instruction set used is almost compatible with IBM 360/370 series instruction set) may be put from virtual memory into a savefile. This savefile may be reformatted by a PASCAL program so as to be submitted to the system linkage utility routines.
The Compiler does some localized optimizations with the aim of producing a compiler suitable for the compilation of application programs. The result is that the code produced seems to be much faster than the code produced by the standard Fortran compiler.
The compilation speed is rather fast averaging 4o lines per second on a 4004/151 and more than 100 lines per second on a 7000/7.755.
The Compiler supports the language standard PASCAL.
File handling is fully implemented by the sequential file access method. Work will be done to support also the (direct access) indexed sequential file access method.
The predicate packed of arrays ; records is ignored as it would not change much on a byte machine.
The procedure dispose is replaced - as in all P-Compilers - by the procedures mark and release.
Global labels may only be used to get back to the main program.
There are no limitations imposed by the compiler.
Additional standard procedures are provided to make operating system services available with the aim to make the compiler suitable for the compilation of system programs.
There is the possibility to interact with the operating system by calls of additional standard procedures.

The system seems to be as efficient and reliable as PASCAL systems are usually.

There is a users manual - written in german language.
For the conditions of availability contact the author.

The Compiler has been developed on the basis of the ETH P4 Compiler. This Compiler has been extended to process full standard PASCAL with some typical modifications (i.e. mark/release, case ... else, variable string assignments and comparisons). The character code is EBCDIC, the setsize is 256 allowing for set of char. The code generation is done on the basis of the intermediate P-code at the end of each procedure trying to do some local optimizations. The code is generated into virtual memory and may be executed immediatly or put into a standard module library.

For further information contact

Dr. Manfred Sommer
SIEMENS AG
Department D AP GE
Postbox 70 00 78
D - 8000 M u n i c h
(West Germany)

The efficiency of SIEMENS PASCAL BS2000

In N. Wirths: "Programming languages...." (Berichte des Instituts für Informatik der ETH Zürich Nr. 17) there is a list of programs for comparativestudies . These programs are measured on a CDC 6400 SCOPE 3.4 installation, assumed to be roughly equivalent to a 370/155 by a remark in the same paper. This set of programs was run for comparison on a SIEMENS 7.755 under BS2000 operating system, assumed to be roughly equivalent to a 370/155 in turn.

Results are:

	CDC 6400 SCOPE 3.4	SIEMENS 7.755 BS2000 V 3.0
1. Powers of two	0.813	0.883
2. Palindromes	2.695	5.223
3. Quicksort (different test data-intsize)	2.861	3.985
4. characount (micro seconds per char)	68	82
5. numericio a) input	1.238	2.541
b) output	0.980	2.260
6. Queens	0679	1.009
7. Prim	1.061	1.083
8. ancestor a) build matrix	0.291	0.267
b) evaluate ancestors	1.667	1.569
c) output matrix	0.578	0.614
9. ancestor-S a) build matrix	?	0.084
setsize = 100 b) evaluate ancestors	?	0.322
c) output matrix	?	0.627

Programs 1,4,6,7,8 indicate that the times used are indeed roughly equivalent; 3,9 are not comparable; the different values on program 5 are probably due to a different file structure; and program 2 is assumed to be an example for the term "roughly equivalent" - it is not known why it behaves different from program 7.

It should be noted that BS2000 is a virtual memory operating system and paging interrupts lead to different execution times of the same program in the order of 10 %.

On the other hand there are still some final optimisations in the code generator not yet implemented - it is hoped that the times will be better by a an order up to 20 % as soon as those optimisations are ready.

The compilation of the compiler yields a performance of 90 lines / second.

There have been some tests on the length of the sequence of instructions for calling "Ackermann". The SIEMENS compiler produces 15 instructions needing 52 bytes of code.

Texas Instruments TI-ASC

Douglas S. Johnson, Advanced Software Technology Dept., M.S. 295, Texas Instruments, Dallas, Texas 75222, tells us that a superset of Pascal called PDL is implemented on the TI-ASC. Through other sources we have learned that PDL was developed using a Pascal cross-compiler running on a Control Data 7600 which produced code for the ASC. PDL was developed for a Ballistic Missile Defense Agency project, and is described in the article: "An extendable approach to computer-aided software requirements engineering" by T.E. Bell, D.C. Bixler, and M.E. Dyer, IEEE Transactions on Software Engineering 3 (Jan., 1977), pp 49-60.

Texas Instruments TI-990, TI-9900

Douglas Johnson (above) also reports that there is a Pascal cross-compiler which runs on an IBM 370 and produces code for the TI-990 and the TI-9910. Several people have told us that TI has developed a native-code compiler which runs on the 990/10 under the DX10 operating system.

A very different implementation for the TI-9900 (a 16-bit micro), MICROPASCAL, is notable for being a stand-alone turnkey Pascal machine with bundled software and hardware. In addition to the materials printed here, the implementors sent us a fairly good-sized manual, mostly in German. Deviations from standard Pascal appear to be: files, with and goto statements, label declarations, and procedures/functions as parameters are not supported. Sets of 64 characters are supported.

We present ourselves:

MICROPASCAL

1.) the implementors are:
H. Schauer, R. Nagler, A. Szer; Institut für Informationssysteme
1040 Wien, Argentinierstraße 8, Austria, Tel. 65 87 31/313
the distributors are:
ECO-Computer GesmbH&Co Kg (Fa. Langschwert)
1010 Wien, Tuchlauben 14, Austria, Tel. 63 35 80

2.) our implementation is called MICROPASCAL

3.) the minimal hardware configuration is the microprocessor TI 9900/4 (Texas Instruments), a mark-sense card-reader and a line-printer (with interfaces). You need no operating system to run the compiler.

4.) only the whole system is selled(hardware and software) and costs 200.000.- ÖS (Austrian Schilling).(about 1500 US $).

5.) the system will be ready for sale in summer 77, we intend to make more of it and we would like to accept bug reports.

6.) documentation is available in form of a supplement to the PASCAL-Report

7.) it fully implements Standard PASCAL beside a few little things caused by the hardware configuration (see documentation).

8.) it is a portable compiler-interpreter system which saves memory and is very slow compared with other systems;it is written in PASCAL and machine-code, 3000 source lines, 12kROM words, no external memory .

9.) the reliability of the system is excellent

10.)it was written in PASCAL and bootstrapped to the microprocessor. it takes three month to implement it on any microprocessor with no special experience of the implementors.

MICROPASCAL is a system that permits the translation and execution of PASCAL programs on a microprocessor. It consists of a microprocessor, memory for the operating system and the user programs and two interfaces for input and output. The main purpose of the system is to support programming education.

Basic concepts: the compiler translates the source program into an intermediate language represented as a tree, where each node represents one declaration and each leave consists of the intermediate code of a PASCAL block in reversed polish notation. This tree is the static information of the program. The compilation does not exceed the level of syntactic decomposition defined by the syntax diagrams in the PASCAL report. At execution time the code is interpreted by aid of the runtime stack which provides the dynamic information. The runtime stack consists of parameters and local data of all the active subroutines. The interpreter performs all context-sensitive checking at the execution time. The intermediate language is compressed by using a numeric code of variable length to represent the identifiers: those which are frequently used are represented by short numbers. Since any information concerning the identifiers is stored in the nodes of the tree, the intermediate code is not redundant. The interpreter is "microprogrammed", i.e. in the intermediate code all operators are calls of subroutines of the interpreter.

Features of the system:

- it supports portability: the machine-independent parts of the system, i.e. the compiler and part of the interpreter are in the intermediate language (and interpreted themselves). Only the nucleus of the interpreter (organisation of the runtime stack and the execution of operations) is machine-dependent and therefore handcoded.

- extremely low requirement of storage (12K ROM): the same interpreter is used to control the compilation, the machine-independent part of the interpretation and the execution of the user program (the problem of runtime efficiency was no constraint to the problem).

- very easy handling: the system is ready as soon power is on. No need for any hardware or software support to provide or maintain a machine-readable program. The input device is a mark-sense card reader and the output is a printed listing.

The machine-independent parts of the system are written in PASCAL and bootstrapped by an existing PASCAL compiler.

UNIVAC 90/70

M. Sommer (see Siemens 4004 announcement, above) responded to Bill Hopkins' request in Newsletter #7 for an implementation for the RCA/Univac Spectra 70: "Stemming from the former cooperation between RCA and SIEMENS there is a close correspondence between SPECTRA 70 and SIEMENS 4004 computers. Our operating system is derived from VMOS - now called BS2000. Our PASCAL implementation is running on a 4004/151 (compatible with SPECTRA 70/61) under BS2000 (compatible with VMOS)." (Letter to B. Hopkins, dated Feb. 2, 1977.)

(* Thanks, Manfred! *)

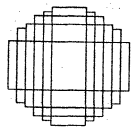
UNIVAC 1100 SERIES

Bill Barabash of SUNY Stony Brook reports that they are in possession of all three Pascal compilers for the U1110. They use the DIKU compiler by Steensgaard-Madsen for beginning students because it only requires 42K. They use the Mike Ball San Diego compiler (60K) in advanced courses because it allows the creation of modules with independent global areas. They also run a preliminary version of the Fischer-LeBlanc Wisconsin compiler which requires 80K and must itself be compiled by Mike Ball's compiler. It's extensive checking appears to be quite sound according to Bill.

VARIAN V-70 SERIES

In a note dated Feb. 1, 1977, Gregory L. Hopwood, Varian Data Machines, 2722 Michelson Drive, Irvine, California 92664, (714/833-2400) states "Yes - we are interested in Pascal. Varian has a Pascal compiler (Brinch Hansen) which runs on our V70 line of minis."

In a letter dated Feb. 4, 1977, Michael Teener, Data Sciences Division, Technology Service Corporation, 2811 Wilshire Boulevard, Santa Monica, California 90403, (213/829-7411) reports:



Technology Service Corporation

Data Sciences Division

2811 Wilshire Boulevard, Santa Monica, California 90403 Phone: (213) 829-7411

4 February 1977

Mr. Andy Mickel
University Computer Center
227 Experimental Engineering Building
University of Minnesota
Minneapolis, Minnesota 55455

Dear Andy:

For the past year or so I have been looking for a Pascal compiler for our Varian V-76 minicomputer. I looked into using a Pascal-P implementation, but that turned out to be too much work to do singlehanded. I mentioned this little project to our local Varian rep, who then shocked me by saying, "But we already have Pascal."

Simply put, anyone can get Pascal from the Varian Users Group (VOICE). The required equipment is a Varian V-70 with 32K+ memory, memory map, Vortex II O.S., extended instruction set and 512 words of writable control store (WCS). This last requirement is of considerable interest

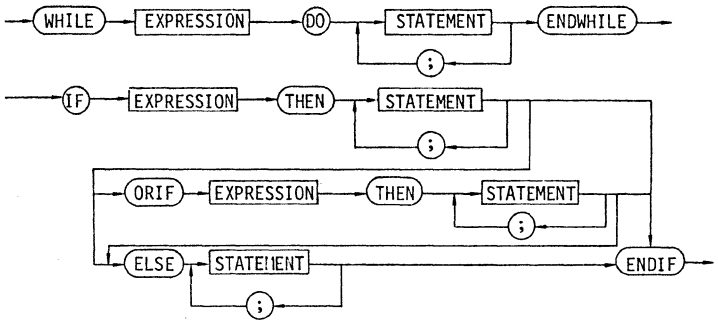
since Varian uses the WCS to set up the V-70 as a Pascal machine...its machine language looks suspiciously like P-code. The compiler itself is quite fast. According to my friends at Varian, it compiles over 1000 statements a minute. Some other characteristics are:

- I/O is not standard, instead it is oriented around Vortex II I/O macros. All files must be opened before using, with reference to files via logical unit numbers. 'GET' and 'PUT' do buffered I/O and 'READ' and 'WRITE' do character by character I/O.
- Programs can be overlaid.
- The range on integers is -32768..32767.
- Integer case labels must be in the range 0...127.
- The range of reals is about $-10^{38}..10^{38}$.
- The relative precision of a real is about 10^{-16}.
- A string must have an even number of characters. (All arrays of type 'CHAR' are packed).
- Enumeration types ('X=(A,B,...)') cannot be defined within record types.
- An enumeration type used as a tag field type can have at most 16 constant identifiers.
- Integer variant labels must be in the range 0..15.
- A set of integers can only include members in the range 0..127 (strangely enough, this is room for all ASCII characters).
- There is no 'text' type.
- Comments are enclosed in double quotes (").
- Brackets '[' and ']' are represented by '(.' and '.)'.
- The horizontal arrow character (underline on newer printers) can be used in identifiers.
- The first ten characters of an identifier are significant.

I haven't had a chance to play with it much, but the programmers at Varian claim it is extremely bug-free for a brand-new compiler. Anyway, anyone can get it from Varian as VOICE #183C8.

As for Pascal itself, I would like to add my voice to the growing crowd of real-world (i.e., non-academic) programmers who would like, or rather, demand formatted and structured I/O. Michael Hagerty's comments in #6 on this subject are excellent.

Aside from I/O and dynamic array parameters (about which enough has been said), I really don't like the 'begin-end' blocking of Pascal. It just doesn't read very well and adds needless confusion to the source code. I would far prefer to use an implicit structure more like Algol 68 or IFTRAN. As a matter of fact, Nancy Brooks of General Research Corporation is implementing a Pascal pre-processor much like IFTRAN (which is a joint GRC-TSC Fortran Pre-Processor) which has the following syntax:



- Similarly for 'FOR', 'CASE', and 'WITH'. (The 'REPEAT' form is already consistent with this.)

The idea is to get rid of all those 'END's. Our experience with IFTRAN leads us to believe that providing unique ending delimiters for compound statements within each type of control structure catches many of the common structural errors in complex programs. The 'IF' - 'ORIF' - 'ELSE' - 'ENDIF' structure is particularly good for this purpose. Besides all that, the resulting pretty-printed listings are a delight to read.

Anyway, Pascal is the best overall language yet, and if the I/O problems are fixed, it could be near perfect for our use.

Keep up the good work.

Sincerely,

Michael Teener
Manager
Computing Center

MT:cs

P.S. Oh yes, Varian Pascal does not have label types or 'GO TO's. How's that for a restriction?

(* Editor's note: we made a mistake! We mistook the commentary on Pascal in this letter to be an explanation of extensions to the Varian implementation. Half of this letter, therefore should have appeared in the Open Forum section. *)

ZILOG Z-80

Ken Bowles has announced an implementation for the Z-80 to be distributed sometime this summer. For more details see the Digital Equipment PDP-11 section of this Newsletter.

According to Jim C. Warren, editor of Dr. Dobbs Journal of Computer Calisthenics & Orthodontia (Oct., 1976 issue, p 6), Niel Colvin of Technical Design Labs, Trenton, New Jersey, has adapted a P-code compiler for the Z-80. The P-code interpreter reportedly occupies about 1K bytes. Another Zilog rumor is that Dean Brown is the person at Zilog to see about Pascal.

Index to Implementation Notices (issues #5 - #8)