# Pascal News

(formerly Pascal Newsletter)

NUMBERS 9 and 10 (combined issue)

Communications about the programming language Pascal by Pascalers

SEPTEMBER, 1977

## Table of Contents

0  POLICY: Pascal News
1  ALL PURPOSE COUPON
3  EDITOR'S CONTRIBUTION
4  HERE AND THERE
   News
   Conferences
   Books and Articles
11  Past Issues of Pascal Newsletter
11  PUG Finances
12  Roster

39  ARTICLES
39  "Pascal at Sydney University"
    - A. J. Gerber and C. C. Morgan
40  "Disposing of Dispose"
    - Stephen P. Wagstaff
42  "What is a Textfile?"
    - William C. Price
43  "Generic Routines and Variable Types in Pascal"
    - B. Austermuehl and H.-J. Hoffmann

47  OPEN FORUM FOR MEMBERS
54  Special Topic: Micro/Personal Computers and Pascal
58  Special Topic: Pascal Standards

60  IMPLEMENTATION NOTES
60  Checklist
60  General Information
61  Software Writing Tools
61  Portable Pascals
63  Pascal Variants
64  Feature Implementation Notes
73  Machine Dependent Implementations

113  POLICY: Pascal User's Group
POLICY: Pascal News (77/09/01)

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

Pascal News contains all we (the editors) know about Pascal; we use it as the vehicle to answer all inquiries because our physical energy and resources for answering individual requests are finite. As PUG grows, we unfortunately succumb to the reality of (1) having to insist that people who need to know "about Pascal" join PUG and read Pascal News - that is why we spend time to produce it! and (2) refusing to return phone calls or answer letters full of questions - we will pass the questions on to the readership of Pascal News. Please understand what the collective effect of individual inquiries has at the "concentrators" (our phones and mailboxes). We are trying honestly to say: "we cannot promise more than we can do."

* An attempt is made to produce Pascal News 4 times during an academic year from July 1 to June 30; usually September, November, February, and May.

* ALL THE NEWS THAT FITS, WE PRINT. Please send written material for Pascal News single spaced and in camera-ready form. Use lines 18.5 cm wide!

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

* Pascal News is divided into flexible sections:

  POLICY - tries to explain the way we do things (ALL PURPOSE COUPON, etc.).
  EDITOR'S CONTRIBUTION - passes along the opinion and point of view of the editor together with changes in the mechanics of PUG operation, etc.
  HERE AND THERE WITH PASCAL - presents news from people, conference announcements and reports, new books and articles (including reviews), notices of Pascal applications, history, membership rosters, etc.
  ARTICLES - contains formal, submitted contributions (such as Pascal philosophy, use of Pascal as a teaching tool, use of Pascal at different computer installations, how to promote Pascal, etc.
  OPEN FORUM FOR MEMBERS - contains short, informal correspondence among members which is of interest to the readership of Pascal News.
  IMPLEMENTATION NOTES - reports news of Pascal implementations: contacts for maintainers, implementors, distributors, and documentors of various implementations as well as where to send bug reports. Qualitative and quantitative descriptions and comparisons of various implementations are publicized. Sections contain information about Software Writing Tools for a Pascal environment, Portable Pascals, Pascal Variants, Feature Implementation Notes, Machine Dependent Implementations, etc.

* Volunteer editors are:
  Andy Mickel - editor
  Tim Bonham and Jim Miner - Implementation Notes editors
  Sara Graffunder - Here and There editor
  John Strait and John Easton - Tasks editors
  David Barron and Rich Stevens - Books and Articles editors
  Rich Cichelli - Software Tools and Applications editor
  George Richmond - past editor (issues 1 through 4)
Pascal User's Group, c/o Andy Mickel  
University Computer Center: 227 EX  
208 SE Union Street  
University of Minnesota  
Minneapolis, MN 55455 USA

// Please enter me as a new member of the PASCAL USER'S GROUP for ___ Academic year(s) ending June 30 ___. I shall receive all 4 issues of Pascal News for each year. Enclosed please find ___ ($4.00 for each year). (* When joining from overseas, check the Pascal News POLICY section on the reverse side for a PUG "regional representative." *)

// Please renew my membership in PASCAL USER'S GROUP for ___ Academic year(s) ending June 30 ___. Enclosed please find ___ ($4.00 for each year).

// Please send a copy of Pascal News Number(s) ___. (* See the Pascal News POLICY section on the reverse side for prices and issues available. *)

// My new address is printed below. Please use it from now on. I'll enclose an old mailing label if I can find one.

// You messed up my address. See below.

// Enclosed please find a contribution (such as what we are doing with Pascal at our computer installation), idea, article, or opinion which I wish to submit for publication in the next issue of Pascal News. (* Please send bug reports to the maintainer of the appropriate implementation listed in the Pascal News IMPLEMENTATION NOTES section. *)

// None of the above.

Other comments:  
From: name ____________________________  
mailing address ____________________________  
phone ____________________________  
computer system(s) ____________________________  
date ____________________________  

(* Your phone number aids communication with other PUG members. * )
JOINING PASCAL USER'S GROUP?
- membership is open to anyone: particularly the Pascal user, teacher, maintainer, implementor, distributor, or just plain fan. Memberships from libraries are also encouraged.
- please enclose the proper prepayment - we will not bill you.
- please do not send us purchase orders - we cannot endure the paper work! (if you are trying to get your organization to pay for your membership, think of the cost of paperwork involved for such a small sum as a PUG membership).
- when you join PUG anytime within an academic year: July 1 to June 30, you will receive all issues of Pascal News for that year unless you request otherwise. You will receive a membership receipt.
- please remember that PUG is run by volunteers who don't consider themselves in the "publishing business." We consider production of Pascal News as simply a means toward the end of promoting Pascal and communicating news of events surrounding Pascal to persons interested in Pascal. We are simply interested in the news ourselves and prefer to share it through Pascal News (rather than having to answer individually every letter and phone call). We desire to keep paperwork to a minimum because we have other work to do.

JOINING THROUGH "REGIONAL REPRESENTATIVES"?
- anyone can join through PUG(USA) - address on reverse side. International telephone: 1-612-376-7290. PUG(USA) produces Pascal News and keeps all mailing addresses on a common list. Regional representatives collect memberships as a service and reprint and distribute Pascal News using mailing labels sent from PUG(USA) which speeds up delivery overseas.

<table>
<thead>
<tr>
<th>European Region (Europe, North Africa, Middle and Near East):</th>
<th>Australasian Region (Australia, New Zealand, Indonesia, Malaysia):</th>
</tr>
</thead>
<tbody>
<tr>
<td>send £2.50 to: Pascal Users' Group (UK) c/o Computer Studies Group Mathematics Department The University Southampton S09 5N Hol telephone: 44-703-559122 x700</td>
<td></td>
</tr>
<tr>
<td>c/o Arthur Sale Dept. of Information Science University of Tasmania GPO Box 252C Hobart, Tasmania 7001 Australia</td>
<td></td>
</tr>
<tr>
<td>telephone: 23 0561</td>
<td></td>
</tr>
</tbody>
</table>

RENEWING?
- please renew early (before August) and please write us a line or two to tell us what you are doing with Pascal, and tell us what you think of PUG and Pascal News to help keep us honest. To save PUG postage, we do not send receipts when you renew.

ORDERING BACK ISSUES OR EXTRA ISSUES?
- Our unusual policy of automatically sending all issues of Pascal News to anyone who joins within an academic year (July 1 to June 30) means that we eliminate many requests for backissues ahead of time, and we don't have to reprint important information in every issue - especially about Pascal implementations!
- Issues 1, 2, 3, and 4 (January, 1974 - August, 1976) are out of print.
- Issues 5, 6, 7, and 8 (September, 1976 - May, 1977):
  - Less than 40 copies each remain at PUG(USA) available for $2 each.
  - Less than 20 copies each remain at PUG(UK) available for £1 each or £2.50 for 6,7,8.
  - None available at PUG(AUS): write to PUG(USA) or PUG(UK).
- Extra single copies of new issues are $2 each - PUG(USA); £1 each - PUG(UK); and $A3 each - PUG(AUS).

SENDING MATERIAL FOR PUBLICATION?
(such as ideas, queries, articles, letters, opinions, notices, news, implementation information, conference announcements and reports, etc.) "ALL THE NEWS THAT FITS, WE PRINT." Please send written material for Pascal News single spaced and in camera-ready form. Use lines 18.5 cm wide! Remember: ALL LETTERS TO US WILL BE PRINTED UNLESS THEY CONTAIN A REQUEST TO THE CONTRARY.

MISCELLANEOUS INQUIRIES? Please remember we will use Pascal News as the vehicle to answer all inquiries and regret to be unable to answer individual requests.
Editor's Contribution

Here is another potpourri of topics:

Pascal Newsletter #8

"Green on green" was not our idea (neither was the thick paper - it destroyed our poverty image). It was a giant disappointment to have worked so hard on #8 and see it come out this way. We agree with the 20 or so people who gently suggested that "we say it in black and white." We were faced with wasting paper and making the newsletter 3 weeks late if we reprinted, or sending it out. We sent it out and were reimbursed by the printer for the extra postage and heavy paper costs. PUGN #1 was mailed from the UK was over 2 months late due to circumstances beyond their control, so it was black on white!

Pascal Jobs

Who says you can't get a job "in the real world" using Pascal? Herb Rubenstein, the first research assistant to work for us at the University Computer Center who learned Pascal before he learned FORTRAN, picked Colorado as a place to live when he graduated with a B. Sci. in Computer Science from the University of Minnesota and then he began job hunting. In 2 months he landed a job with a rapidly growing engineering peripherals firm, AutoTrol, and is working almost exclusively with Pascal.

Also see the OPEN FORUM section for a letter from Neil Barta.

New Australasian Distribution Center for PUG

To solve problems with slow mail to Australia (as well as currency exchange), Arthur Sale, prolific PUGN contributor at the University of Tasmania, has kindly set up a distribution center this summer (winter) much like Judy Mullins and David Barron did for Europe a year ago.

The area served is Australia, New Zealand, Indonesia, and Malaysia. We at PUG(USA) are confused about why the price is so high; apparently we were to receive a letter from Arthur over 2 months ago with the details, but it was lost in the mail. Other details are on the reverse side of the ALL PURPOSE COUPON.

Computer Companies Using Pascal

It is past time to print a list we've been keeping of computer companies who are seriously using Pascal. This is so we can argue back that "Pascal is being used for serious real world work" when accused otherwise!

Total conversion internally to the company:

- Texas Instruments, January, 1977 ("from micros to super computers")
- Harris Data Communications, March, 1977 ("Pascal is our language - replacing FORTRAN and COBOL" - Tom Spurrier.)
- Companies Using Pascal for future software systems:
  - Cray Research (CRAY-2)
  - Control Data Corporation (Cyber 270 series) (They have already been using it for the 2550 and the Cyber 18)
- DATA 100 Corporation (model 78)
- Companies marketing Pascal as a user product:
  - Honeywell; Computer Automation; Four Phase Systems; Varian Data Machines (Sperry Univac).

New Developments - Micro/Personal Computers

Several PUG members took my request seriously to write to several of the personal computing journals to promote Pascal over BASIC (see Editor's Contribution PUGN #8). David Mundle, George Cohn, and Tim Bonham have written letters. At Frank Brewster's and Rich Cichelli's urging, I sent personal letters and a free copy of #8 (the only free copies we had given) to the editors of 14 computing journals. We received warm responses from half a dozen.

Also we've been getting new members from their readership, some who are so curious to know about Pascal that they are dying to get this issue of PUGN. I'm really encouraged at these developments because these computers represent the future and we have an early start (unlike on the current dinosaur systems).

See the OPEN FORUM section.

Pascal News

We changed the name to avoid confusion by people who think a newsletter is 4 pages long. This issue is a combined one because it contains so much material - and it is also late. We had to revise nearly everything: the cover, the coupon, policy, and do a summary for the implementation notes! This has good side effects because PUGN #2 was late in Europe, and renewals have been slow to come in. Next issue will be in February. Deadline for material is the last day in December: (77/12/31).

New Policies

Look at all the new editors! Please read the revised policy pages on the inside covers (front and back). The major change is that we are declaring that we are tired of processing purchase orders and answering requests for information "about Pascal" from people who won't join PUG and read Pascal News. It may sound strange, but we print everything we know about Pascal in Pascal News.

Back Issues

It is really difficult to plan ahead on back issues with a growing membership. Nevertheless we made it through last year with some extra copies of each issue. But we incurred some tremendous distribution problems which caused unjustified delays in sending back issues to people who joined PUG after mid-February. I apologize, and hope that we have learned enough from our mistakes to do better this year.

Membership

We began collecting PUG memberships on 76/03/03. Here are some interesting membership totals: 317 on 76/09/13 (#5 to press); 368 on 76/09/09 (#4 mailed); 516 on 76/11/14 (#6 to press); 560 on 76/12/10 (#6 mailed); 598 on 76/12/29 (#7 to press); 644 on 77/01/13 (#7 mailed); 943 on 77/04/26 (#8 to press); 984 on 77/06/12 (#8 mailed); 1095 on 77/06/30 (end of year); 1306 on 77/09/07 as I write this (759 active).

We have 211 new members and 560 renewals since 77/07/01 with renewals still rolling in.

PUG Finances

I last printed information in PUGN #6. Last year (our first) we promised and delivered 4 issues of PUG Newsletter. What we did not know was how popular PUG was going to be. We also delivered a few things we did not promise: 230 copies of back issue #4, mass mailings to get to new and old people, letters to implementors to get compiler information and unfortunately slow service to late joiners (sorry, but we wish you had joined earlier).

See the HERE AND THERE section for details under "PUG Finances". We show a small loss almost exactly 15 - and our crude accounting knowledge doesn't account for all the back issues produced with 76-77 money and sold in 77-78 (since July 1., we have sold 243 at PUG(USA) alone. So I claim we did okay.

Andrew - 77/09/07.
Here and There With Pascal

(* Here are extracts from almost all of PUG's mail. To reiterate what we've said elsewhere, many of the inquiries we get are answered in previous issues. If you are a member, please try to find answers to your questions from Pascal news before you write to us. If you aren't a member and you want information that's in an issue that's already out, we'll tell you to join rather than to answer each inquiry with a personal letter. *)

Attn: Centro Ciencias de la Computacion, Universidad Catolica de Chile, Casilla 114 D, Santiago, Chili: "Is there any FULL PASCAL implementation for the IBM 370?" (* 77/6/7 *)

Phil W. Bergstrom, 128 Jackson Ave., Madison, AL 35758: "TRW has a PASCAL program on the CDC 7600 and TAI-ASC with 40000 statements and 1100 procedures, REVS, the Requirements Engineering and Validation System, supporting interactive color graphics, CALCOMP plotting, and a relational data base. We have implemented a complete 7600 PASCAL system." (* 77/6/2 *)

G W Birk Lund, 2250 Copperplinth Square, Reston, VA 22091: "I am presently working on a Pascal computer for the IBM Series 1, and should be finished in September 1977." (* 77/6/2 *)

Kenneth Bowen, P.O. Box 1123, Rancho Santa Fe, CA 92067: "Looks like we will be working with CONDUIT on getting a (Standard) ANSI BASIC running under our PASCAL system. Object: entice Basic users over to PASCAL by making a switch very convenient. This will be the only truly portable BASIC we know of." (* 77/6/22 *)

Bill Brennan, 39 Jody Drive, Norristown, PA 19401: "I am presently engaged in implementing a PASCAL for Sparc/univac 3000 computers - (This activity is for my education mostly, not for release.) I certainly could use the information your newsletter will provide. For your information, I heard of the PASCAL system, and should be finished in September 1977."

Arthur A. Brown, 1101 New Hampshire Ave. NW, Washington, DC 20007: "I am a professional translator of Russian mathematics, and will be glad to abstract the Proceedings of the All-Union Symposium on Implementation Techniques for New Programming Languages. (We sent them off right away, but just received word from Arthur Brown that an English translation has been published as Vol. 47 of Springer-Verlag's Lecture Notes in Computer Science."

(* 77/6/10 *)

Thomas V. Burnet, Computer Center, Dickinson College, Carlisle, PA 17013: "What PASCAL is available for a PDP-11 running RSTS?" (* 77/6/30 *)

Edwin J. Calde, Dept. E152, AAI Corp., P.O. Box 5676, Baltimore, MD 21204: "Would appreciate information concerning the availability of PASCAL for the SEL 8000 series or SEL 32." (* 77/7/19 *)

Patrick Chevaux, DEC, Quai Ernest Ansermet 20, B.P. 23, CH-1211 - Geneva 8, Switzerland: "I am urgently looking for a PASCAL compiler running on PDP-11 under RSX-11M operating system, and I wonder if you know about such a product. If so, could you please give me a few indications about it, as well as the person to contact and perhaps how to obtain it." (* 77/7/11 *)

D. Michael Clarkson, DBMS Research and Development, California Software Products, 525 N. Cabrillo Park Dr., Suite 300, Santa Ana, CA 92701: "My company is currently involved in implementing a lot of high-level translatable system software using PASCAL." (* 77/6/27 *)

Kurt Cockrum, 3398 Utah, Riverside, CA 92507: "K. A. Lovestedt should get in touch with Tom Payne, Math Dept., University of California at Riverside, Riverside, CA 92507 for information on HP-3000 implementations of Pascal. I believe that John Hayward of UCR has written a P-code interpreter that runs on the 3000."

(* 77/7/2 *)

Bill Collins, 3M Co., Bldg. 235-P247, St. Paul, MN 55101: "We are considering using PASCAL as a Systems Implementation Language for microprocessor based systems, using a PDP11 as a host for cross-compilation and system monitoring." (* 77/6/13 *)

Larry Crane, EDS, 1200 Locust, Des Moines, IA 50309: "Thanks for sending us the PUG newsletter, hopefully we'll be able to get hold of something good. If not we'll just have to develop it. With luck we'll have an operating system in Pascal. To the bit bucket with Fortran, even COBOL will be overcome."

(* Response to Andy's letters to personal computing publications has been heartening, if somewhat humorous at times. In Creative Computing, for example, the "Pascal" User's Group was mentioned, but the address got lost in the press. Nonetheless, high school student Steven Trapp, 5020 Mulcare Drive, Columbia Heights, MN 55421, deduced the address from Andy's name and the name of the building and wrote to ask for an all-purpose coupon. *)

Jack Croman, Systems Analyst, USC School of Medicine, Hoffman Res. Ctr., Rm. 805, 2025 Zonal Ave., Los Angeles, CA 90033: "From his letter which we saw in Byte, May 1977. *" "At present, supporting a full blown high level language compiler is quite an achievement for a personal computer; supporting several is out of the question. For this reason it is important to make the best possible selection and to select some obscure educational vernacular such as PASCAL because it is estetrically more pleasing, and [sic] would leave personal computing where it is right now: a lot of hardware with very little software."

(* From his letter which we saw in Byte, May 1977. *"

Kenneth A. Dickey, 1662 Stromberg, Arcata, CA 95521: "I am especially interested in Pascal applications dealing with environmental modeling, approximations, simultaneous equations, and text editing." (* 77/7/11 *)

John Dickinson, Dept. of Elec. Engr., Univ. of Idaho, Moscow, ID 83843: "I also would like to ask your help in locating a good implementation of PASCAL for a IBM 370 machine. I understand there are many such implementations and my question is for which is best for a student environment. I plan to use PASCAL in a beginning computer science class and so I would like a version that is easy to use and one that has clear error messages." (* 77/6/30 *)

Jim Elam, 150 Lombard, No. 601, San Francisco, CA 94111: "I would be interested in information on usage in a production environment and efficiency of generated code on 370 gear!" (* 77/6/2 *)

Gary Feierbach, Advanced Studies Dept., Inst. for Advanced Computation, P.O. Box 9071, Sunnyvale, CA 94086: "We currently have Pascal upon our KI-10 and plan to put it up on several other machines, including a version on the ILLIAC IV."

(* 77/6/26 *)

Charles N. Fliege, Academic Computing Center, Univ. of Wisconsin, 1210 West Dayton Street, Madison, WI 53706: "We have a proposal for PL/I - like varying length strings for you for the next few months - it appears to extend PASCAL fixed length strings rather nicely. Also, I'll be in Minneapolis for an Univac User's Meeting in mid-October. If it's convenient, I may be able to stop by and talk some Pascal with you. (I'll be heading a PASCAL "birds of a feather" session at the meeting)." (* 77/8/30 *)

Dan Fylstra, 22 Weitz St. Charlestown, MA 02134: "To put this letter in context: Dan is an editor/consultant for Byte."

(* Initially I plan to write an article explaining the Pascal Compiler *'}
Here and There With Pascal

features and strengths of Pascal, aimed at the BASIC-oriented beginning programmer or casual user. But I’ll certainly include notes on the status of Pascal implementations and especially their availability on micros (since the news is so good).

"You can invite people to write or call me if they have late-breaking news that deserves a wider audience than the User’s Group itself. Since everyone connected with Byte is enthusiastic about Pascal, articles, new product announcements, and material for "Byte’s Bits" or the "Technical Forum" are always welcome. These should be sent to Byte’s regular address in Peterborough."  (* 7/7/82 *)

Richard Gemeichards, Jr., Discipledata, Inc., 110 S. Downey, Indianapolis, IN 46219: "Please advise if Pascal operates on any NCR hardware—such as NCR Century 201 or NCR Criterion."  (* 4/25/77 *)

James D. George, Computer Branch, Underwater Sound Reference Division, Naval Research Laboratory, Room 8337, Orlando, FL 32886: "The Naval Research Laboratory has several PDP-lls, and is using RSX11M and RSX11D. I would be very much interested in finding out more about PASCAL under RSX11, and would appreciate any leads you could provide."  (* 7/7/5/7 *)

Roger Guiberson, Dept. of Physics, Univ. of Illinois, Urbana, IL 61801: "Even though I know you don’t like it, you can add my name to the list of people who want an OTHERWISE (or whatever) clause added to the CASE statement. I particularly liked George Richmond’s article. I’m not sure with all the things he said, but most of his points seem reasonable. I’m not sure I agree with his point about partial-IR evaluation of boolean expressions. While I’ll admit it will help some problems array indexes and the like I’m finding out that the FTN (* CDC Fortran *) method of logical if evaluation (i.e., convert the whole mess into a logical (or Boolean) result) and subsequent jump on true/false is faster on machines like the [ Cyber] 175 and probably also the 76. Considering the trend toward faster hardware, it may not be a good idea to explicitly demand partial evaluation.

"I agree with Legenhausen’s comment about pushing PASCAL in the appropriate micro computer journals. Maybe the way to do it is to develop a standalone PASCAL compiler for the very basic system with no more memory than 8K (if you must) and then distribute it for a nominal fee—say $10 or $15. And no, I don’t have the time to do it."  (* 7/7/6/6 *)

George E. Hayman, 556 Parker Rd., W. Melbourne, FL 32901: "Many maintainers who arbitrarily change Pascal at their sites are guilty of the NIH (Not Invented Here) syndrome. If I haven’t thought of it then it isn’t any good.

"I’m interested in Sequential Pascal, directly compilable, for the PDP-11, with an RT-11/RSX-11 operating system."  (* 7/7/5/25 *)

Carl Helmers, BYTE Publications, 70 Main St., Peterborough NH 03458: "A couple of comments about the Zilog rumor. All the information came from the same source and later proved premature. At the IEEE Computer Society Asilomar conference this year, a Zilog representative could not confirm Pascal as a programming model for advanced architectures, but hinted strongly that research in the direction of instruction sets optimized for high level languages such as Pascal is being performed. A talk in the lobby of the West Coast Computer Fair’s convention hotel with one of Motorola’s LIS designers strongly hinted of the possibility of built in microcode for language constructs in the next generation of integrated circuits.

A strong suggestion people involved with the implementation of languages should seek out LSI design engineers in order to inject ideas about appropriate features to be built into the designs of future microprocessor products.  (* 7/7/6/20 *)

Richard Hendrickson, Gray Research Inc., 7850 Metro Parkway, Suite 213, Minneapolis, MN 55420: "Keep up good work. Articles like the one by Barron and Mullins in No. 7 will do wonders for the job of keeping FORTRAN and eliminating PASCAL as major computing language."  (* 7/7/5/23 *)

Sam Hills, 3514 Louisiana Ave., Lakw., New Orleans, LA 70125: "I am interested in developing a subset of PASCAL to run on a hobby-type microcomputer such as the Altair or INSAI, and any information you could supply would be greatly appreciated.

Tao-Yang Hsieh, VIDA, 77 Ortega Ave., Mountain View, CA 94040: "I am considering implementing Pascal on our HP2100 system and would appreciate very much if you could assist me in obtaining a copy of Pascal P-code compiler and a copy of Pascal compiler written in Pascal."  (* 7/7/8/1 *)

Jon F. Hurna, Dept. of Information and Comp. Sci., Univ. of Calif., Irvine, CA 92717: "I’m... working for Univac on the side. .. We would find life a whole lot easier if we had a reasonable file comparison program to work with. You wouldn’t happen to know of anyone who’s written one in Pascal, would you? Please let us know."  (* 7/7/12 *)

Alfred J. Hubert, Inhalation Toxicology Res. Inst., P.O. Box 5890, Albuquerque, NM 87115: "We are working with John Barr of Hughes Aircraft to get Brian Lucas’ NBS PASCAL written in PASCAL for RSX-ll users of DEC PDP-11’s (along with real time and character string extensions)"  (* 7/7/6/22 *)

Geoffrey Hunter, Chemistry Dept., York Univ., Downsview M3J 1P3, Ontario, Canada: "Thanks for your memo of 7/7/05/24. I am of course familiar with Pascal and actually taught a course one year using Mirth’s book "Systematic Programming: an Introduction." I used "Algol" rather than PASCAL, Simula, Algol 68, etc. for the Waterloo talk, because it is, as you note, the ancestor of all current structured programming languages.

On first acquaintance I was an enthusiast for PASCAL, but after some practical experience, and after reading Habermann’s article in Acta Informatica Vol. 3 (1973) p. 47, I changed some reservations about some features of the language. Especially the lack of block structure (environment structuring— as distinct from control structures and procedures in particular), and the lack of dynamically dimensioned arrays, are, it seems to me, conceptually overlooked. PASCAL’s strong point is, of course, data structuring.

There is a danger with any organization such as PUG—that it becomes the defender of a fixed particular definition and implementation of the language. Guard against this...

(* 7/7/6/1 *)

Aron K. Insinga, 126 Dupont Hall, University of Delaware, Newark, DE 19711: "I am interested in using Pascal under UNIX (and DEC-supported operating systems) as well as on micro-processors (the LSI-11, Motorola 68000, and Intel 8080, in particular) with compilation and assembly done on the larger PDP-11 system.  (* 7/7/17/9 *)

Milt Jolson, SIRPC, 25 Blegen Hall, Univ. of MN, Minneapolis, MN 55455: "It may interest PUG members to know that the LEAA (Law Enforcement Assistance Administration), a division of the Justice Department, requires, by legally enforceable regulation, that all criminal justice software be in ANSI FORTRAN or ANSI COBOL."  (* 7/7/8/18 *)

Hatti Karinen, and Jyrki Puoci, Compiler Project, Room 2113, Computer Center, Tampere Univ. of Technology, Finland, 527, 33101 Tampero 10, Finland: "We would appreciate information about PUG and the Pascal Newsletter, especially as we have in mind to implement Pascal on our PDP 11/70."  (* 7/7/8/17a *)

Barbara J. Karkut, Box 942, Easton, PA 18042: "An interested in the Pascal compiler for the 2-80 microcomputer."  (* 7/7/6/6 *)

Doug Kaye, Dubt Film Labs Inc., 245 W. 53th Street, New York, NY 10019: "I anxiously await Newsletter #9 with writeups about PASCAL on Data General gear."  (* 7/7/7/21 *)

Ed Keith, Citrus College, 18824 E. Foothill Blvd., Azusa, CA 91702: "Please send data on available systems, assemblers etc. I have a XEROX 560, INSAI 8080, SWIFCONF 6800."  (* 7/7/4/28 *)

Thomas J. Kelly, Jr.: 120 East Street Road, C3-9, Warnminster, PA 18094: "I am interested in obtaining a Pascal compiler for any Burroughs computer; especially for the B5500, B6700, or B7700."  (* 7/7/5/16 *)

PASCAL NEWS #8 4-10 SEPTEMBER 1977 PAGE 11
David Miller, 11203A Avalanche Way, Columbia, MD 21044: "Please sign me up for the PASCAL User's Group. I've been so busy developing PASCAL (relocatable, for DEC 11/45) and an application system, I failed to notice the Group has grown so much. Finally got to reading some SIGPLAN notices, and ran across your letter." (* 77/7/6 *)

Carlton Mills, 203 North Gregory, Urbana, IL 61801: "We are working on Pascal compiler for microprocessors. It is a highly optimized cross compiler running on the B6700 (Burroughs). Currently I am looking for venture capital to get it on the market. I will let you know if we get ready to announce it." (* 77/8/2 *)

J. Marsh, Dept. of Computer Sciences, Painter Hall 3.28, Univ. of Texas at Austin, Austin, TX 78712: "I would appreciate receiving any information about Pascal implementation on NOVA computers.

Our department has recently acquired two NOVA's for which we wish to get the compilers. The size of the compiler would probably make it prohibitive for the NOVA's. If you know of any existing implementation, please send us the information." (* 77/8/29 *)

Tom Oberly, Academic Computing, Grinnell College, Grinnell, IA 50112: "We are looking for a Pascal system which will run on our PDP 11/70 (RTX/E)." (* 77/6/7 *)

Brian Nelson, Computer Sciences, 2801 W. Bancroft Street, U. of Toledo, Toledo, OH 43606: "I am trying to locate a Pascal compiler for use on a PDP 11/70 and a PDP 11/40." (* 77/6/2 *)

John W. Nunnally, Harding College, Box 744, Searcy, AR 72142: "Harding College has just ordered a Pascal compiler from Oregon Museum of Science and Industry (OMSI). It is a modified version of ESL's implementation that is supposed to run under RTX/E Version 68 (with the RT-11 emulator). We will let you know how it goes." (* 77/5/25 *)

Carol Anne Ogden, Software Technique, Inc., 100 Fornander Walk, Alexandria, VA 22314: "I am preparing some material for publication on PASCAL for micros in my capacity as Consulting Editor of Mini-Micro Systems and EDN." (* From a note to PUG member Peter Zechmeister, 77/6/15 *)

Shmuel Pelleg, Computer Science Center, University of Maryland, College Park, MD 20742: "Do you know of any Pascal compilers working under UNIX?" (* 77/8/28 *)

Lee Potter, DARCOM ALMSA, Attn: DRXLTL, P.O. Box 1578, St. Louis, MO 63188: "My agency is planning to try Pascal as a systems implementation language on IBM 360 and several minicomputers implementing architecture. Pascal's main attraction to us now is systems portability." (* 77/9/1 *)

Walter F. Pratsch, Albertinenstrasse 29, D-1000 Berlin 37, Germany: "I would like to mention that I am working in the field of system-simulation (methodology, applications in the field of urban and regional planning). If you know any people using PASCAL for the development of simulation-systems (event-oriented as well as continuous), please let me know their addresses." (* 77/6/10 *)

Bruce K. Ray, Polymorphic Computer Systems, P.O. Box 3581, Boulder, CO 80303: "I am interested in developing a Pascal compiler for use with the NOVA-series computer, and am therefore interested in anything and everything which may help me in the task. Is there a PASCAL written in a mini-PASCAL (subset) which is available which would be easier to bootstrap, and if so, who, how, where, and how much?" (* 77/8/16 *)

Harlan R. Ribble, P.O. Box 3182, Boulder, CO 80303: "I am a graduate student in Computer Science at the University of Colorado working on an implementation of a PASCAL to JANUS compiler. I was informed by someone I met on the CDC PLATO system that I might be able to get some information from you regarding the PASCAL users Group." (* 77/8/19 *)

Bo Röder, AB Atomenergi, Fack, 611 01 Nykoping 1, Sweden: "AB Atomenergi is a research and development center for nuclear and other energy forms. At our data center we have a CDC CYBER 172 with 131 K memory, and NO'S I.2 operating system. We plan to install Pascal on it and hereby apply for membership in Pascal User's Group, as individuals or as an organization, whichever the policy of PUG is." (* 77/8/22 *)
Peter Kuschmeyer, Luitpold-Gymnasium, Seeastrasse 1, D-8000 Munich, Germany: "We got a PDP 11/34 with 64K Word Core, 2 Disks RG5, a LA180 Lineprinter and 7 VT50 screens. RSTS/S Release 68, BASIC."

"We ought to teach Informatics to our pupils aged 10 to 20."

"We would like to get a PASCAL-Compiler, interactive if possible, running in RSTS if possible."

"Can you help us?" (* 77/4/2 *)

Bernie Rosman, Math/CS Dept., Framingham State College, Framingham, MA 01701: "I am trying to get CDC 6000 Pascal 2 for Mass. State College Computer Network (Cyber 72, 73). Currently, we have Pascal release 1 update 11 which has some bugs; e.g., SQRT doesn't work (fixed by MSGCON). Also: we're now using Pascal in data structures and CS II (2nd semester-freshman) courses. We have, however, not yet switched to Pascal in CS 1. Finally, we hope to install Pascal on our new PDP-11/34." (* 7/5/31 *)

David J. Ryksa, Dept. of Computer and Info. Science, 2036 Neil Ave. Mall, Columbus, Ohio 43210: "I am an active user of a DEC-10 version and would like to find other versions and documents for the DEC-10."

(" 77/4/4 )

Carlos Schuel, Depto de Sistemas, Instituto Tecnologico de Monterrey, Monterrey, Mexico: "We would like to have the compiler of the PASCAL system; please mail me back all the information and prices, manuals, etc." (* 7/7/8/5 *)

Barry Searle, Towerc Floor 10C, Transport Canada, Section TASK, Place de Ville, Ottawa, Ontario K1A 0B8, Canada: "The Canadian Dept. of Transport will be converting to Pascal on PDP-11 equipment." (* 7/8/25/5 *)

David Segal, 111 Third Ave. BK, New York, NY 10003: "I am planning to get a microcomputer and would like to implement something more useful than BASIC for it. I am interested in Pascal and am curious about its implementation and its use."

(" 7/8/3 ")

Bruce Seiler, UCLA Dept. of Chemistry, Los Angeles, CA 90024: "I am interested in the implementation of PASCAL on microprocessor based systems." (* 7/5/23 *)

Michael Settle, 751 Washington, No. 115, Arlington, TX 76011: "I have a confession to make--I don't have any idea what PASCAL is. I work with the huge Inland Business Monsters and tinker with my own Altair. There has been so much discussion of BCPL and Pascal in terms of data structures and lower case version of Pascal."

(" 7/8/15 ")

David Elliot Shaw, Structured Systems Corp., 343 Second St. Suite K, Los Altos, CA 94022: "You are performing a welcome service for the community of Pascal users, implementors, fans. . . .

On the accompanying sheet we describe (as compactly as possible) the STRUCTURED SYSTEMS PASCAL-SS compiler for the PDP-11." (*) 7/7/12 *)

Jeffrey G. Shaw, P.O. Box 2678, Menlo Park, CA 94025: "Could you direct me to an individual or group that might have a Pascal compiler for the 8080 or 280 micros?"

(* 7/7/8/18 *)

Evan L. Solley, The Life Support Systems Group, Ltd., 2432 NW Johnson, Portland, OR 97210: "... Also enclosed is a write-up and sample listing for a PASCAL cross-reference we developed some time ago. It is an extension of Wirth's PCKRF, which, we are sure, is more reliable. Its symbols tables are currently set up to process ESI Pascal (V5.5) for RT-11, but can be easily modified for use with other compilers."

(" The program is licensed and distributed in ASCII source form for a fee of $25.00. Distributed media include magtape (9-track 8000 bpi), DECtape, RG05 cartridge, or card deck (800 cards). Media should be provided by Licensees. RT-11 users will additionally receive a special executable version, with CSI and GCL interface to version 3 of RT-11 and LSOE's RT-11X extension of version 2C." (* 7/4/23 *)

Tom Spruill, Electronics Systems Division, Harris Corp., P.O. Box 37, Melbourne, Fl 32901: "Harris Corp. headquarters has issued a corporate directive that Pascal is our language. There are over 100 computer centers in the corporation. It will be used for system level development initially and then in applications areas." (* 7/7/6/2 *)

John P. Stallings, Tynshare, Corporate Offices, 20705 Valley Green Drive, Cupertino, CA 95014: "Once again I find myself potentially involved in a project concerning Pascal and have decided that it is past time for me to associate myself with an appropriate source of information."

"Could you tell me how to go about joining the Pascal User's Group, and if possible, how to obtain a list of available Pascal compilers for the PDP-11?" (* 7/7/18 *)

Rod Steel, MS 60-450, Tektronix Inc., P.O. Box 300, Beaverton, OR 97007: "I have a partially debugged version of Mike Ball and Co.'s Concurrent Pascal cross-compiler for the Interdata 7/16 running on our DEC KL10 (translated from Sequential Pascal to the lower case version of PASREL)."

(* 7/7/5/31 *)

W. Richard Stevens, Kitt Peak National Observatory, P.O. Box 26732, Tuscon, AZ 85726: "What follows is extracted from an article Richard wrote for the Kitt Peak Computing Newsletter."

"The PASCAL language, because of its features designed into it, has the ability to detect programming errors that would be undetected by any FORTRAN system. I have personally found that this feature alone cuts in half the time needed to develop a new program." (* 7/1/3 *)

"The article mentions other features of PASCAL which make it useful at Kitt Peak.

(* 77/1/3 - The article mentions other features of PASCAL which make it useful at Kitt Peak."

Peter Summer, Interdata Computers Pty. Ltd., 30 Kings Park Rd., West Perth, Western Australia: "I have personally found that this feature alone cuts in half the time needed to develop a new program."

(* 77/5/23 *)

Markku Suni, Computer Centre, Univ. of Turku, SF-20500 Turku 50, Finland: "Since I am interested in experimenting with our own Pascal compiler and have spent some nice time on it, I would like to join in . . . . We here have a PDP-11 with KA processor, 128k of core, 2 RP03 discs, one T10 mag tape unit, card reader, line printer, and usual sort of terminals." (* 7/7/4/8 *)

Rodney Thayer, Central Research Group, P.O. Box 451, Harvard, MA 01451: "A few people in my area (myself included) are investigating R. E. Berry's G. of Lancaster Pascal for the Data General NOSA. If I am closer than England for somebody, they are welcome to write to me to find out about Lancaster PASCAL." (* 7/7/7/7 *)

Mike Tillier, 2501 N. Lancaster Ln. No. 178, Plymouth, MN 55441: "Interested in Pascal for NOSA/ECLIPS." (* 7/7/14 *)

Martin Tuori, Behavioral Sci. Div., Defense and Civil Inst. of Environmental Medicine, P.O. Box 2000, Downers, Ontario, M3M 389, Canada: "We will be running ESI Pascal under RSX1M, as soon as ESI has it ready." (* 7/7/7/26 *)

Univ. of Texas at: Austin (The statistics from their newsletter indicate that Pascal and Pascal 2 accounted for 5% of their total use in March 1977.)

James A. Veltena, System Development, Data 100 Corp., Box 1222, Minneapolis, MN 55440: (* He reports that there is a class in Pascal at Data 100. Ten to fifteen people were enrolled. Nine memberships came from people at the company. *)

Kenneth R. Wadland, Computer Science Dept., Fitchburg State College, Fitchburg, MA 01440: "Although I have not used PASCAL much, I have become quite interested in it from talking to Professor Bergeron of the University of New Hampshire. He has been modifying a DEC System-10 compiler written in Germany."

"I intend to teach PASCAL in my Data Structures course and later in my Systems Programming course on a DEC Cyber 72. As a teaching device, I think it is far superior to any of the other standard languages." (* 7/7/6/29 *)
German Chapter of the ACM, a meeting on Pascal.

(* This is rather late notice, but we'll hope that interested members will at least be able to attend the conference, if not submit papers. *) Meeting October 14, 1977 in Kaiserslautern. Papers will include such subjects as "Implementations," "Pascal in Schools," "Applications," and "Pascal and Microprocessors." For more information get in contact with G. Nees, German Chapter of the ACM, c/o Siemens AG, E 54, Mozartstr. 33/b, D-6750 Kaiserslautern, Germany. (* Our thanks to Hans-Wilhelm Wipperman for keeping us informed about the conference. We hope to have a report from the conference in No. 11. *)

Pascal Day or Pascal Workshop, McMaster Univ., Hamilton, Ontario, Canada.

(* From a letter from Nick Solntseff *) I am starting to plan a "Pascal Day" or a "Pascal Workshop" to be held at McMaster on March 3, 1978. I will be getting in touch with the Regional ACM group and the IEEE Computer Society, to see if they want to sponsor it. I am thinking of asking for brief reports on implementations, use of Pascal for teaching, etc. (* For more information, write to Nick Solntseff, Dept. of Applied Mathematics, 1280 Main St. West, Hamilton, Ontario, Canada L8S 4K1; or call (416) 525-9140. *)


(* Thanks for this report to Nick Solntseff *) I did not have too much interest shown at IFIP in a meeting of PUG, but I am really surprised as it was almost impossible to get in touch with people one knew were at the conference.

"The computerized message system was terrible to say the least, but anyone interested should have seen my manual notice on the general notice board.

"In all, I gathered nine people over coffee in the hospitality lounge at various times, but decided that a more formal meeting was not called for."

Meeting of the Pascal sub-group, APCEI, Nice, France, June 13-14, 1977.

(* PUG member Olivier Lecarme, IMAN, University of Nice, Parc Valrose, F-06034 - Nice CEDEX, France, has sent us a bulletin, which he publishes regularly before meetings of the sub-group, of articles to serve as a basis of discussion for the meeting of the sub-group. We'll try to get word to you in advance of the next meeting, but in the meantime, if you wish to receive the bulletin and/or be notified of meetings, write to Olivier Lecarme. *)

Titles of Articles:

"The language Pascal as support for teaching introductory programming," R. Rousseau.

"The future of Pascal (extensions and standardization)," Andy Nickal.

"Some tools for users of Pascal at Rennes," l'equipe Simone.

"Simulator of machines in Pascal," D. Thalmann.

"Pascal/CII-iris 80 and 10070," P. Maurice.


BOOKS AND ARTICLES

We've had no news from David Barron. Rich Stevens supplied us with one item. George Richardson's bibliography, which we didn't have room for in No. 8, appears separately. A price list for some formerly out-of-print documentation appears under IMPLEMENTATIONS.

LANGUAGES


(* From the publisher's blurb *) "... detailed handbook showing you how to develop simple and reliable operating systems from scratch using Concurrent Pascal."

"Proceedings of the All-Union Symposium on Implementation Techniques for New Programming Languages," Novosibirsk 1975. English translation published by Springer-Verlag as Volume 47 of their Lecture Notes in Computer Science. (* PUG member Arthur Brown, who had offered to abstract the Russian, sent us news of the English translation in lieu of the abstract. We'll try to get more information for No. 11.

TEXTBOOKS

(* A Summary of all known Pascal textbooks, partly reprinted from newsletters 5-8 *)

Atwood, J. W., Standard Pascal, to be published. (* Note: we haven't heard anything new about this book. For more information, write to J. W. Atwood, Dept. of Comp. Sci., Sir George Williams Campus, Concordia Univ., Montreal, Quebec, Canada H3G 1M8. *)


An introduction to Pascal for non-programmers which in spite of its length fails to cover any data structures besides arrays. A rewrite of a book based on PL/I which still carries the smell of FL/I-a foreward stating the contrary notwithstanding.
A complete introduction to Pascal for non-programmers using an interactive graphics approach and the keller teaching method.

A rewrite of a book by the same name on PL/I.

Schneider, G. Michael, Steven W. Weingart, and David M. Perlman, *Introduction to Programming and Problem Solving with PASCAL*, New York: Wiley, to be published in January 1978. A camera-ready copy of the manuscript can be obtained by writing Gene Davyport, Editor, John Wiley and Sons Publisher, 605 Third Avenue, New York, NY 10016. The manuscript may, with written permission, be duplicated for class use until the publication date.
A complete introduction to Pascal for computer science majors.

Webster, C.A.G., *Introduction to Pascal*, Heyden, 1976. $11.00, 5.50, DM35.00.
A book for beginning computer science majors which received a bad review in Pascal Newsletter No. 8 because, among other things, there are numerous errors and the old language definition was used.

(* From the preface *) "A book which introduces programming as the art or technique of formulating algorithms in a systematic manner, recognizing that it is a discipline in its own right." (* This introductory book only covers Pascal through arrays *)

(* From the cover *) "... lucid, systematic, and penetrating treatment of basic and dynamic data structures, sorting, recursive algorithms, language structures, and compilers."

IMPLEMENTATIONS

Price list on Reports of Interest--hard-to-get implementation information:

Through the courtesy of George H. Richmond and his co-workers Karin Bruce and Michele Dowd, reprints of some hard-to-get Pascal documentation is now available. Write to:

Karin and Michele--Pascal Distribution
Computing Center Library: 3645 Marine St.
Univ. of Colorado,
Boulder, CO 80309
or call (303) 492-8131.
(* These all can be ordered from North America at the price listed. All others must include overseas postage. *)

(* Includes an entire listing of a Pascal-S compiler/interpreter in Pascal. *)

(* Description of the internal design and performance of Pascal-6000 *)

(* Describes the portable Pascal compiler and interpreter. *)

(* Letter received from David Barrow ~ 7/7/25 *)

"I am sorry I have not been able to write earlier with news of the publication of the Proceedings of the Pascal Symposium. We had hoped that these would appear in the Springer-Verlag 'Lecture Notes in Computer Science,' but after an initially favorable reaction Springar delayed, and have finally declined to publish. However, I am pleased to be able to report that Wiley-Interscience have agreed in principle to publish the proceedings. I am currently discussing details with them, and hope to be able to give you firm details very shortly."

APPLICATIONS


Algorithms expressed in Pascal.

A description of how to carry attributes of computation such as temperature, energy, fuel consumption, etc., and units expressing these attributes such as celsius, kelvin, joules, liters per km with numerical quantities used in scientific and engineering problems. This circumvents problems which arise in dealing only with pure (dimension-less) real numbers in current programming languages.


Algorithms expressed in Pascal.

Suggests that new machines introduced by semiconductor manufacturers may be called "Pascal machines" instead of stack machines because the Pascal compiler generates code for a 'hypothetical stack machine' and manufacturers may start building machines, using LSI technology, like the hypothetical one.


Algorithms expressed in Pascal.

A larger description of the pretty printer announced as available for distribution in Pascal Newsletter 6, page 70. (* This and the other article below from the July issue were drawn to our attention by PUG member Harry M. Murphy. *)

 Mentions Pascal as one high-level language used on small computers, and urges readers to be aware of it.

Algorithms in Pascal.

A set of guidelines for standard naming, formatting and commenting conventions in Pascal programs and why programmers should adhere to them.

A report on the National Computer Conference, which lists Pascal as a programming language available on personal computers, but which says that there are few implementations of it so far.

(* From the summary. *) "Two language design methods based on principles derived from the denotation approach to programming language semantics are described and illustrated by an application to the language Pascal. The principles are, firstly, the correspondence between parametric and declarative mechanisms, and secondly, a principle of abstraction for programming languages adapted from set theory. Several useful extensions and generalizations of Pascal emerge by applying these principles, including a solution to the array parameter problem, and a modularization facility."
PAST ISSUES OF Pascal Newsletter (now Pascal News)

George Richmond, Computing Center, University of Colorado, started Pascal Newsletter with issue #1 in January, 1974. He proceeded to produce 3 more issues while doing the other thankless chores of distributing 2 Pascal compilers to dozens of sites and promoting Pascal in other ways.

In mid-1975 John Strait and I proposed a Pascal User's Group after having talked to several other Pascalers around the U.S. At the Minneapolis ACM '75 conference in October, 1975, we launched the group at an ad hoc meeting (35 persons) convened by Rich Cichelli and Bob (Warren) Johnson. A year later we began the task of producing 4 issues of Pascal Newsletter which PUG as a group assumed responsibility for.

John and I edited the first 2 issues with help from Tim Bonham on the Implementations section. By issue #8 John had less time for the constant demands of the newsletter and only promised occasional help, but with #8 Jim Miner, Sara Graffunder, and others volunteered to help. With this issue (#9 & #10), we have spread the load quite a bit, which only causes coordination problems!

#1 January, 1974, University of Colorado Computing Center, (also SIGPLAN Notices 9:3 1974 March) 8 pages, edited by George Richmond. (* Mostly contains descriptions of the CDC-6000 implementation of unrevised Pascal.*) out of print

#2 May, 1974, University of Colorado Computing Center, (also SIGPLAN Notices 9:11 1974 November) 18 pages, edited by George Richmond. (* A Pascal history; news of other implementations for unrevised Pascal; news of the new CDC-6000 implementation for revised Pascal.*) out of print

#3 February, 1975, University of Colorado Computing Center, (also SIGPLAN Notices 11:2 1975 February) 19 pages, edited by George Richmond. (* Announcement of the book: Pascal User Manual and Report; Pascal usage questionnaire; revised History of Pascal; bibliography; news of Pascal-P; more on Pascal-6000 for CDC machines; letters to the editor.*) out of print

#4 August, 1975, University of Colorado Computing Center, 103 pages (103 numbered pages), edited by George Richmond. (* 36 letters of correspondence dealing mostly with various implementations; Implementors list; bibliography; news of new release of Pascal-P.*) out of print

#5 September, 1975, Pascal User's Group, University of Minnesota Computer Center, 124 pages (65 numbered pages), edited by Andy Mickel. (* Short notes, 5 articles, general correspondence, and implementation notes were featured; Christian Jacobi, ETH Zurich supplied a description of Dynamic Array Parameters.*)

#6 November, 1975, Pascal User's Group, University of Minnesota Computer Center, 180 pages (91 numbered pages), edited by Andy Mickel. (* News from members; a full membership roster; conference notices; information on back issues; 6 articles including 2 proposing directions for Pascal by G. Michael Schneider of the U of Minnesota, and Rich Cichelli of Lehigh University; much implementation news.*)

#7 February, 1976, Pascal User's Group, University of Minnesota Computer Center, 86 pages (45 numbered pages), edited by Andy Mickel. (* More News from members; books; 3 articles; correspondence; implementation notes.*)

#8 May, 1976, Pascal User's Group, University of Minnesota Computer Center, 124 pages (65 numbered pages), edited by Andy Mickel. (* News from members; Conferences; Books; Applications; 6 articles including one by Ken Bowles about a very complete inexpensive implementation for nearly every microprocessor in existence; Special topic: officials standardization and clarified definition of Pascal; Portable Pascals; Feature implementation notes; Machine Dependent implementations, Index.*)

Back issue ordering information for #5-#8 is on the back of the ALL PURPOSE COUPON.

PUG FINANCES 1976-1977

Here are the details for our finances last academic year by both PUG(USA) and PUG(UK). For additional information see the EDITOR'S CONTRIBUTION (for a real con) under "PUG Finances."

PUG(USA) Accounts:

Income:
$3980.00 995 memberships @ $4 (76-77)
161.73 contributions
70.00 miscellaneous back issues sold @ $1

$4211.73 Total Income.

Expenditure:
$123.37 buying (230) and mailing #4 from George Richmond
492.70 printing (700) and mailing #4
1239.50 printing (1050) and mailing #6
697.17 printing (1000) and mailing #7
1071.60 printing (1000) and mailing #8
30.23 mailing originals of 5-8, etc. to PUG(UK) for reprinting
92.13 promoting PUG (mass mailings)
10.00 refunds for overpayment
19.00 backissue requests for #4 forwarded to George Richmond
101.09 miscellaneous postage for automatic backissues

$3876.79 TOTAL Expenditure.

PUG(UK) Accounts: (submitted by David Barron, 15 August, 1977)

Income:
£249.90 subscriptions for 76-77 (99 @ 2.50; 1 @ 2.40)

£249.20 TOTAL Income.

Expenditure:
£70.86 printing 250 copies of No. 6
29.14 printing 450 copies of No. 7
105.34 printing 450 copies of No. 8
171.06 postage for 6, 7, and 8 including back issues

£376.40 total production costs
90.01 printing and posting No. 5 (450 copies)

£466.41 TOTAL Expenditure.

PUG(USA) surplus = $334.94
PUG(UK) deficit = £216.51 = $380.00 approx.
Total deficit for year = $ 46.06

Andy Mickel 77/09/01
You can see at a glance who is at a well known organization at a well known place. The roster makes a great organizing tool for our mutual communication! Please look yourself up to check for accuracy and then you can see who is nearby; why not phone them and talk about Pascal?

States with over 50 PUG members are: California - 171; Minnesota - 128; Texas - 62; Massachusetts - 61; and countries: United Kingdom - 101; Canada - 59; Germany - 57.
1. Introduction

The temptation to "play" with software is more often than not too great to resist, and we succumbed. Our experience (over 12 months) with our changes has given us confidence in them, and only a few of our original mods have been retracted. We are also pleased to add that users seem more than happy to use these "extra" features, and that we often have to turn away requests for the more fanciful changes which do get proposed from time to time. We do realize that Pascal is deliberately designed to be minimal, efficiently implementable and so on. We also have some rather strong views on where the absence of certain features actually hinders many programmers, as opposed to those features which are genuinely used rarely by a small group of users (which does not include ourselves).

1. Implementation-dependent/oriented features

1.1 KRONOS-oriented changes

Necessary as always. In particular, allowing INPUT & OUTPUT to operate interactively under TELEX; and letting the compiler accept line numbers (sequence numbers) on source files.

1.2 Listing format

A listing format modelled after the 1972 Stanford Algol W implementation was adopted. Particularly valuable is the ability for programmers to check BEGIN-END, etc., nesting with level indicators on the left-hand side of the listing. %-cards (lines with a '%' in the first column) may be used to control the listing's spacing, titling, paging, etc. A more interesting %-card is the "$INCLUDE <filename>" which allows source text from other files to be interspersed within the main source file.

1.3 For the benefit of student users

Some more checking facilities have been added. There is a compile-time check against the assignment of a value to a for-loop control variable. The $%T+ option has the added effect of initializing the stack at run-time to 0. This allows hardware checking for undefined reals and (most) pointers. An oversight in the first column) may be used to control the listing's spacing, titling, paging, etc. A more interesting %-card is the "$INCLUDE <filename>" which allows source text from other files to be interspersed within the main source file.

1.4 Fieldlength handling

A different FL-handling discipline is implemented. The user may preset his run-time FL at compile-time by use of the ($FLXXXXX) control comment. This has the effect of forcing the program (at run-time) to grab an amount of core equal to its code space, plus "xxxx". The default setting is:


These settings may, of course, always be overridden at run-time by not running in REDUC mode.

1.5 Glitter

a. A dayfile message has been added to provide timing information.

b. An option ($%W+) provides warnings if any of the language extensions detailed below are used by the programmer. The default setting is "on".

2. Language-oriented extensions

Most of the language "extensions" detailed below do not, we believe, run contrary to the "spirit" of Pascal. They were all implemented quite cheaply and with little or no effect on compiler efficiency. Our experience with them has vindicated them at least as far as we are concerned.

2.1 Reading strings

The standard procedure "read" is extended so that variables of type "string" (i.e. packed arrays of char) may be read. The definition is as follows: if f is a textfile, and s a string variable, then "read(f,s)" is equivalent to:

begin
  a[i] := f[i];
  if not eoln(f) then get(f)
end;

Note that read(f,s) never does a "get(f)" when eoln(f) is true. Hence it never causes a "readln(f)", and in addition it right-fills incompletely-read strings with blanks.

2.2 Reading and writing symbolic scalar types

It is possible in our version to read and write symbolic scalar variables (RED, GREEN, BLUE, ...). Packages of form CAT, DOG, MOUSE; ... TRUE, FALSE etc.) This allows the language to be more generous in its treatment of scalar variables - most users complain of the absence of this feature at one time or another. An additional benefit is that the post-mortem dump can now really dump such variables.

2.3 Case-statement revamp

These extensions are arguably at odds with Pascal's "minimal language" philosophy, but turn out to be incredibly useful. They are: (i) the addition to the case-statement label list of a constant "range", and (ii) the addition of a "default" label. The first of these is surprisingly absent from standard Pascal in view of the recent addition of constant ranges in the syntax of sets (e.g. [1..3]). We have a sneaking suspicion that this was not implemented because the Pascal-6000 lexical analyser maps colon (:) and dot-dot (.) into the same internal symbol, thus making compilation of things like

```
case 1 of
  3,...,12: begin ....
```

rather awkward for a one-symbol-lookahead compiler. Our (ad hoc) solution was to use the word symbol to in place of "." here. The default label is represented by else and is executed if no constant satisfies the evaluation of the case-expression. A typical example is:

```
case ch of
  'A' to 'Z': ....;
  'a' to 'z': ....;
  else: ....
end;
```
2.4 And for systems programmers

Two further modifications were made to the language which are not intended for use by "general-purpose programmers". They enable one to undertake systems programming from within Pascal exclusively. The extensions in this regard allow one to treat pointers as integers (and vice-versa), and to access the address of a variable. They are:

1. The "pointer to" operator. The use of "t" is extended so that if "<variable>" has been declared thus:
   
   ```pascal
   var <variable> : <type>
   ```

   then the value of the expression "t<variable>" is a pointer to <variable>, and is of type "t<type>".

2. The mechanism provided by the standard functions "ord" and "nchr" is extended in the following way: every type declaration allows the use of a corresponding "type-function" throughout the scope of its declaration. The type-function is of one argument, of any type; the function-result is the same argument (bit-for-bit), but with its type changed to that of the type-function.

3. In conclusion

We would like to stress that our changes to Pascal-6000 have not detracted from the overall efficiency of the compiler or its object programs. Our experience over the past year or two with these changes has definitely vindicated them, and we feel they are worth the consideration of the Pascal community at large.

(* Received (77/01/03)*)

Tony Gerber and Carroll Morgan are at the Basser Dept. of Computer Science University of Sydney *

### Abstract

This paper presents an argument for an automatic garbage collection system for dynamic variables in PASCAL, obviating the need for, and risks associated with, user-controlled de-allocation (e.g. DISPOSE). It also describes how complete protection from "dangling" pointers may be obtained.

Keywords: Protection, pointer, garbage collection, dynamic variables, PASCAL.

---

### Introduction

Consider the following PASCAL code fragment:

```pascal
type T = record
  x : integer;
  ...
end;

var P, Q : T;

new(P);
Q := P;
... dispose(P);
Q.x := 1;
```

The space occupied by the variable Q has been de-allocated and yet Q has a non-nil value. This problem is mentioned in [1] and discussed in [2]. I should like to propose a solution which uses a garbage collection system based on the block structure of PASCAL. Performing the garbage collection is simple and inexpensive, and the programmer can easily arrange matters so that the space occupied by dynamic variables is not allocated for any significantly longer time than that for which the variable is actually required.

### The Scope of a Dynamic Variable

Consider

```pascal
procedure OUTER;
  type T = ...;
  var P1 : T;
procedure INNER;
  var P2 : T;
  ...
```

Variables of type T cannot exist outside the scope of OUTER, and neither can pointers of type T. Thus, whenever a dynamic variable is created, the space it occupies can be maintained on a list associated with the appropriate "procedure-instance" (or in implementation terms, "stack frame").
On exit from any procedure, the whole list can then quite simply be returned to the allocation system.

The programmer can minimise his storage expenses by giving type declarations the minimum possible scope (which is good programming anyway). However, the question remains: what happens in the case where pointers reference identical structures but with differing type identifiers (and hence, possibly, differing scopes)? It seems reasonable to regard pointers as referencing types rather than structures, and whenever two types have the same structure, to regard this as a "coincidence". This gives the programmer a fine degree of control over both the lifetime and accessibility of dynamic variables. Thus, with

```pascal
type T = ...;

procedure OUTER;
  type T1 = t;
  var P1 : tT1;
  procedure INNER;
    type T2 = T;
    var P2 : tT2;
    new(P2);
    P1 := P2;
```

the distinction in the programmers mind between types T, T1 and T2 would be recognised and the final statement would be flagged as an error by the compiler, as an incompatible assignment.

Associated Protection Measures

Should it be desired to trap all possible address violations associated with pointer variables, four accompanying measures are required.

Firstly, to ensure that spurious pointer values do not exist, all pointers should be given an initial value of nil.

Secondly, (assuming that pointers are implemented as main memory addresses!), external files should not be allowed to contain components of type pointer.

The third and fourth points concern variant records.

When dealing with access to the variant part of a record (static or dynamic), the compiler should generate code to perform a run-time check that the value of the tag-field is consistent with the variant implied (this check could perhaps be optional in general but mandatory for components of type pointer).

Finally, if variants are overlaid, there is a possibility that a dynamic change of variant would result in erroneous access to memory space beyond that occupied by the variable. This can be dealt with either by forcing all variants to be specified with NEW and disallowing any further assignments to tag-fields or by disallowing the "variant" form of NEW so that the maximum required space is always allocated (The latter would allow dynamic changes of variant).

The last two points are discussed in detail in [2].

Summary

By incorporating an automatic garbage collection system for dynamic variables in PASCAL, together with appropriate scope rules for type identifiers, the responsibility for de-allocation can be taken away from the user, and hence a class of potential address violation errors can be eliminated. Given a little programmer awareness, the cost of this added protection need not be significant. Together with the other protection measures noted all address violation errors can be either prevented at compile time or immediately trapped at run time.

References

1. Wirth, N. Pascal Newsletter No. 5 September '76 p.29

(* Received 77/05/17 *)
What is a Textfile?

The PASCAL revised report, section 6.2.4 in particular, is in serious error as to the nature of textfiles. This error arises -- or is demonstrated by -- the definition of type TEXT as FILE OF CHAR. (As a typographical convention, program fragments are presented in upper case, and the pointer operator, up-arrow, is represented by the character @). As a result of this lapse, complex special-case notions are introduced as primitive concepts. Please notice that I am not advocating a change in the language, or an abolition of existing notation: I merely propose a new, more useful, understanding and definition of the textfile notion.

First, consider the files F and G:

F: TEXT;
G: FILE OF FILE OF CHAR.

Obviously, a READ or WRITE performed on F will perform the same on G, the "inner" file in G. Some of the auxiliary I/O constructs, however, change in a very enlightening fashion: reviewing all available literature on the semantics of PASCAL file operations, we conclude that

**WRITE(F) -> PUT(H).**
**READ(F) -> BEGIN WHILE NOT EOF(H) DO GET(H): GET(G) END, and**
**EOLN(F) -> EOF(H).**

We conclude that to supply the structure implied by WRITE, READ, and EOLN, a textfile is at least a file of lines, where each line is a file of characters.

There is even more to a textfile: we haven't considered the PAGE statement. Let's add another declaration:

H: FILE OF FILE OF FILE OF CHAR.

Now, anywhere we used G, we can use H: logically, however, the re-representation of READ changes. The whole set of equivalent construct-pairs becomes, with the addition of the PAGE statement:

**READ(F) -> READ(H).**
**WRITE(F) -> WRITE(H).**
**READLN(F) -> BEGIN WHILE NOT EOF(H) DO GET(H): IF NOT EOF(H) THEN IF EOF(H) THEN GET(H) ELSE GET(H) END,**

At this point, we have developed the structure that is necessary and sufficient to support all the standard textfile operations. As an added benefit (or is it a side effect?) we have a better appreciation of the embedded file, or file-of-file, concept. Before running off to reimplement textfiles the new way into your favorite compiler, however, let's give some thought to extensibility.

If a textfile is considered as merely a nest of files, then those implementations which would like to give access to such things as page numbers, line numbers, and vertical printer spacing ("carriage control") will have to kludge those features in as primitives: thus we would be back where we started. If, however, we consider TEXT to be predeclared as follows, we notice some nice hooks:

**TYPE TEXT: RECORD**

(*EXTERNAL FILE NAME, ETC*)

P:FILE OF RECORD
(*PAGE NUMBER*)

L:FILE OF RECORD
(*LINENUMBER*)
(*SPACING*)

C:FILE OF CHAR END

The comments point out places where interesting implementation-dependent features can reside.

William C. Price
28282 SW Mountain Rd.
West Linn, OR 97068 USA

(* Received 77/06/10 *)

WCP:pt
Generic routines and variable types in PASCAL

01/27/77

Abstract

Generic routines and variable types, as introduced in ELL [1], are means to postpone the binding time of routines and data. In this paper it is examined to what degree such features may be carried over to PASCAL without severe violation of the static type checking requirement. We conclude that generic routines fit to PASCAL, while variable types have to be subject to strong restrictions. Besides, they may be used only in connection with a special syntactic form.

Introduction

This paper is concerned with the possibility of extending PASCAL by two main features of Ben Wegbreit’s language ELL [1], namely “generic routines” and “treatment of data types as values.”

In a generic routine in the sense of ELL, formal parameters are bound to a set of data types, and the type of an argument must be an element of the type set of the corresponding formal. Inside a generic different actions may be executed depending on argument types. Thus, a generic routine may be regarded as a collection of different routines for arguments of different types under a single name, i.e., as the abstraction of an operation that requires different algorithms for different input data types.

From the second feature of ELL, the treatment of types as values, follows the ability to evaluate and compute types and therefore the existence of type variables and type functions. Types are not treated statically, but in a dynamic environment, hence, variable types, too, form abstractions, since routines are not to be bound to their data at definition time, but the structure of objects may become known only at compile or even run time.

We are aware of the difference between ELL, which is an interpreter-based language (based on the facilities of the ECL system) where type checking may be deferred until run time, and PASCAL, where all types have to be known to the compiler. Our goal is to determine the restrictions to be posed on the ELL features that are used to postpone the binding of procedures and data from programming time to compile or even run time by the type checking requirements of PASCAL.

In this paper, we deal with features of ELL in PASCAL terminology, too, so we speak of types instead of modes and ignore that ELL is an expression language, i.e., we distinguish between statements and expressions. The extended PASCAL that we investigate is referred to as PASCAL-ELL.

A more detailed discussion (in German) of the ideas and results may be found in [2]. An experimental version of the proposed extensions is implemented based on the PASCAL P2-compiler.

Principles of the PASCAL extension

The PASCAL design principles reliability and clarity of the language are the criteria for the extension. These principles, in the extreme, require static type checking and prohibit run-time type checking of operands. In PASCAL, the compiler is able to assert the compatibility of operand types for each operation, including field selection and array subscripts. Therefore, in our extension we have to add static information about variable types to the compiler whenever we are able to, if we fail, as a consequence, there must exist interfaces to fix variable types at compile time. At those interfaces, however, we have to admit dynamic type checks to ensure the validity of the fix at run time, and there type-dependent run-time errors may occur if the run-time instance of the type is not in the set of allowed types. These interfaces must be the only points where dynamic checks are required, and the user must be aware of run-time errors only at those points.

Union types and the generic form in ELL

In ELL, we find union types. The meaning of “union”, there, is only the postponement of type choice, i.e., at run time each object and variable has during its lifetime a definite and unchangeable type. In particular, the definite type of a union-type parameter is determined by the argument type and cannot be changed subsequent to creation.

A generic routine has parameters of union types. Inside its body, the alternatives of the union types may be singled out by means of the “generic form” that resembles a case statement in PASCAL. A generic form consists of several alternative branches and a header naming the parameters the types of which are to be fixed. The right-hand sides of the branches are statements, the left-hand sides are formed of type-lists (one entry for each generic parameter) and additional (optional) predicates. In the type-lists, alternatives (or unions of alternatives) of the parameters’ union types are specified, to which the types of the corresponding parameters are fixed inside the branches.

The appropriate branch for a given combination of argument types may be selected at compile time, if all types in one of the type-lists “cover” (for definition of cover see [1]), the corresponding argument types. Since argument types may be unions (if arguments are parameters of other routines), an argument type may be only partially covered (LL), or it is a type-list element, i.e., some alternatives of the argument type are not alternatives of the type-list element, while others are. Then the compiler is not able to decide whether the definite run-time type fits or not, and must

Department of Computer Science
Darmstadt University of Technology
Pascal News #9
Page 47
October 1977

Pascal News

Generic routines and variable types in PASCAL

01/27/77

Abstract

Generic routines and variable types, as introduced in ELL [1], are means to postpone the binding time of routines and data. In this paper it is examined to what degree such features may be carried over to PASCAL without severe violation of the static type checking requirement. We conclude that generic routines fit to PASCAL, while variable types have to be subject to strong restrictions. Besides, they may be used only in connection with a special syntactic form.

Introduction

This paper is concerned with the possibility of extending PASCAL by two main features of Ben Wegbreit’s language ELL [1], namely “generic routines” and “treatment of data types as values.”

In a generic routine in the sense of ELL, formal parameters are bound to a set of data types, and the type of an argument must be an element of the type set of the corresponding formal. Inside a generic different actions may be executed depending on argument types. Thus, a generic routine may be regarded as a collection of different routines for arguments of different types under a single name, i.e., as the abstraction of an operation that requires different algorithms for different input data types.

From the second feature of ELL, the treatment of types as values, follows the ability to evaluate and compute types and therefore the existence of type variables and type functions. Types are not treated statically, but in a dynamic environment, hence, variable types, too, form abstractions, since routines are not to be bound to their data at definition time, but the structure of objects may become known only at compile or even run time.

We are aware of the difference between ELL, which is an interpreter-based language (based on the facilities of the ECL system) where type checking may be deferred until run time, and PASCAL, where all types have to be known to the compiler. Our goal is to determine the restrictions to be posed on the ELL features that are used to postpone the binding of procedures and data from programming time to compile or even run time by the type checking requirements of PASCAL.

In this paper, we deal with features of ELL in PASCAL terminology, too, so we speak of types instead of modes and ignore that ELL is an expression language, i.e., we distinguish between statements and expressions. The extended PASCAL that we investigate is referred to as PASCAL-ELL.

A more detailed discussion (in German) of the ideas and results may be found in [2]. An experimental version of the proposed extensions is implemented based on the PASCAL P2-compiler.

Principles of the PASCAL extension

The PASCAL design principles reliability and clarity of the language are the criteria for the extension. These principles, in the extreme, require static type checking and prohibit run-time type checking of operands. In PASCAL, the compiler is able to assert the compatibility of operand types for each operation, including field selection and array subscripts. Therefore, in our extension we have to add static information about variable types to the compiler whenever we are able to, if we fail, as a consequence, there must exist interfaces to fix variable types at compile time. At those interfaces, however, we have to admit dynamic type checks to ensure the validity of the fix at run time, and there type-dependent run-time errors may occur if the run-time instance of the type is not in the set of allowed types. These interfaces must be the only points where dynamic checks are required, and the user must be aware of run-time errors only at those points.

Union types and the generic form in ELL

In ELL, we find union types. The meaning of “union”, there, is only the postponement of type choice, i.e., at run time each object and variable has during its lifetime a definite and unchangeable type. In particular, the definite type of a union-type parameter is determined by the argument type and cannot be changed subsequent to creation.

A generic routine has parameters of union types. Inside its body, the alternatives of the union types may be singled out by means of the “generic form” that resembles a case statement in PASCAL. A generic form consists of several alternative branches and a header naming the parameters the types of which are to be fixed. The right-hand sides of the branches are statements, the left-hand sides are formed of type-lists (one entry for each generic parameter) and additional (optional) predicates. In the type-lists, alternatives (or unions of alternatives) of the parameters’ union types are specified, to which the types of the corresponding parameters are fixed inside the branches.

The appropriate branch for a given combination of argument types may be selected at compile time, if all types in one of the type-lists “cover” (for definition of cover see [1]), the corresponding argument types. Since argument types may be unions (if arguments are parameters of other routines), an argument type may be only partially covered (LL), or it is a type-list element, i.e., some alternatives of the argument type are not alternatives of the type-list element, while others are. Then the compiler is not able to decide whether the definite run-time type fits or not, and must
Generic routines and variable types in PASCAL

07/12/77/720

generate a run time test. This holds, too, if the additional predicate is not evaluable at compile time.

The PASCAL-UFL generic extension

In Neubreit's ECL system there exists a compiler as well as an interpreter, both fully compatible. Each may call the other as a subroutine. Therefore the compiler is able to evaluate parts of a compilation unit (routine) and use the value instead of the formula, so predicates of a generic form may be evaluated by the compiler and a compile-time selection may be done. In the generic form carried over to PASCAL-UFL, predicates are not allowed. There are two reasons for this decision:

1. We have no interpreter in our system, therefore, predicates are not evaluable at compile-time and a run time selection is necessary for each call of a generic routine with predicates, even if the covering of all argument types is asserted, so the number of possible run time errors increases.

2. By design, a decision in a generic form is a decision depending on the types, not on the values of the arguments. Accordingly, predicates in a generic form should be predicates on types only. The type classes that are defined implicitly by predicates, however, do not have such a specific structure that the compiler is able to handle them (e.g. all one-dimensional arrays). The compiler will not be able to determine any component statically. Therefore a static type checking will be possible inside the branch and so the advantages of the generic form will be lost.

Union types in the sense of ECL however, fit to the requirements of PASCAL. First, the structure is that of the compiler, so there is no difference to a normal PASCAL type after the selection of one single alternative in a generic form. Second, the type constant during the lifetime of a union-type parameter allows a stack implementation of such parameters, when the procedure is entered, the definite type with its length becomes known (since this happens at run time, run time type descriptors have to be generated by the compiler). Since the length is unchangeable, an address on the stack may be computed for each union-type parameter and the argument values may be copied. Access to the parameter values is indirect via the compile-time-computable local address, where a pointer to the run-time-computable real address is to be stored. Since we are able to put union-type parameters on the stack as opposed to the PASCAL heap (where flexible-type-length parameters would have to be put), there are no problems with RELEASE commands of the user. So union-type parameters do not mess up PASCAL's storage management scheme.

The demand for static type checking implies that each generic parameter is fixed to a defined, compiler-known type (including a union type) at the entry into a generic branch. If that type is a union, operations on the parameter are restricted to assignments and tests of equality inside the static type checking requires fully fixed operand types for any other operation. This restriction forces a programmer to write down repeatedly similar branches for similar, but different types (e.g. array of integer vs. array of real). The PASCAL convention of identifying a type by name, not by its structure, disallows us to define arrays of unions and so to handle similar structures in a single branch, since we then need to have variables of type array of unions. Union-typed variables, however, will not be allowed, since (a) each variable must have an unchangeable type (there is no chance of a postselection of types at assignment with parameters) and (b) untyped variables would impose further need for run time type checks. So the disadvantage of multiple writing down can not be remedied by using unions, we will see later that there is a slight improvement by use of variable types.

With the given restrictions, the generic form is easily transferable to PASCAL. Thus PASCAL-UFL procedure body may be either a normal PASCAL procedure or a generic form. The only violation of static type checking by the use of a generic form may occur if only partial covering is given at compile time and thus a run time check is needed for branch selection. If at run time the combination of argument types does not fit to any of the branches, a type-dependent run time error will result, violating the principle of static type checking. Thus, however, only occurs at a well-defined interface, where the user may expect it. Besides, the number of run time branch selections will normally be small.

Example:

begin

procedure p (var a: integer; var b: char; var c: integer);
begin
  a := a + b;
  b := c;
  c := a;
end;

end.

Types as values

In PASCAL, types are static descriptions of the structure of a class of objects. In ECL, however, type generators are callable functions and deliver a type value. The compiler evaluates such functions under assistance of the interpreter. Consequently, user-defined type functions as well as type variables are permitted. If a type function is not evaluable at compile time, a call to the interpreter is generated, i.e. type checking is delegated to run time. Type variables may be "frozen", i.e. evaluated in an interpreted environment of a compilation step, and their value may be used as a type constant in the compilation unit. "Unfrozen" type variables, again, require type checking at run time. The facilities of evaluating type functions and freezing type variables enables the compiler to feed down variable types. The binding of routines to types is transferred from compile time to run time, but an interpreted environment has to be be involved in this process.
Generic Routines and Variable Types in PASCAL

In PASCAL, we do not have the facility to freeze variables, since there is no interpreted environment available. A variable type at programming time remains variable at compile time (although being fixed at run time). Static type checking, however, requires a wide range of constancy for type variables, since these have to act as representatives of the run time types: two variables declared by the same type variable must have the same run time type, since the compiler can check their types only by means of the name of the type variable. As a consequence, a type variable in PASCAL-GVT must not be declared in its defining unit in any other than any other than (1) recursive data structures (2) similar structures over different base types by one definition, as to the static type checking. However, this does not mean such a dynamic structuring, since the compiler is not able to determine the depth of the recursion statically and so cannot provide access to any component. This implies that one cannot define operations on objects declared by recursive type functions. So recursion

must be forbidden. In addition, even without recursion, type functions are not compile time evaluable because of the existence of parameters and globals. Since the compiler is unable to determine the structure of any function-defined type, those types are obviously meaningless and thus are forbidden in PASCAL-GVT.

As to other type-valued expressions, we must consider the above remark on recursion, here the same holds for iteration. If we allow complex type-valued expressions, it will always be possible to assign them to array (...) of TVAR, which structure will not be recognizable statically. So we must forbid, too, complex type-valued expressions and allow only type variables and type constants to appear on the right hand side of an assignment statement.

Structures Over Variable Types

---

Structures (arrays, records) over unions cannot be defined, since union type is only allowed for parameter specification; parameters in PASCAL, however, must be specified not by a type structure, but by a type name, and compatibility of actual and formal parameters is determined only by equality of the type name, not by similarity of structure (or a certain kind of covering, if unions were involved). Since we do not allow variables of union type, there cannot exist any compatible argument for a formal type "structure of union".

Structures over type variables, however, are meaningful, since type variables may be used in any context where other types may be used. Although its overall structure is known to the compiler, the entire variable of such a type is considered a variable-type variable. Insofar as its real address is determined only at run time and access is indirect, records are physically restructured to shift variable-type fields to the end and so to give information about the relative address of the fixed-typed fields to the compiler, both arrays and records may be stored continuously without any use of internal pointers and thus copying may be done without examining substructures. Since the compiler has information about the overall structure of such an array or record, only component types, if variable, have to be fixed in a generic expression.

Examples:

---

VAR TVAR: MODULE;
PROCEDURE P (IPAR: MODULE);
TYPE VARRAY = ARRAY (...) OF TVAR;
VARREC = RECORD VFELD: TVAR;
VARVFELD: VARRAY;
VFELD: INTEGER
END;
Generic Routinen and variable types in PASCAL

var VV: TVAR; J: INTEGER;

begin
\[ \text{I is generic (VV) of (INTEGER)} : VV; \]
\[ \text{REAL} : \text{TRUNC} (VV) \]
end;

Conclusions

Type variability is a means of separating data and programs. In ELL, it is a meaningful instrument, since compile and run time are homogenous in that compile time of routines and run time of type evaluation may be the same. Thus, library routines may be written data independent and their types evaluated in an interpreted environment of their compilation, so achieving an efficient and type-secure object code in spite of data independence, since type checking may be done at compile time.

In PASCAL, where compile and run time are strictly separated, the static type checking mechanism imposes strong restrictions onto the use of type variables, making them constant in hierarchically ordered procedures, we are not able to extend PASCAL's type scheme by iterator types and type functions, which may be regarded as classes of types, since these two features require dynamic treatment. Thus the type definition part remains the only place where types may be constructed.

The use of type variables in PASCAL-GVT is along two axes:

1. Directly for declaration; this is an extension of the generic parameter mechanism to variables;
2. As base type of a structure; we may look at similar structures under one single type, if we represent the different base types by a type variable. However, if we want to abstract from the base type of a structure and use a type variable, we have to copy each instance of the structure with variable base type to an instance of the structure with a fixed base type, since each instance of the structure with variable base type, even if we were able to fix that variable base type in a generic expression to the right of variable base type, the PASCAL convention of considering two equal structures as different types enforces copying. Such a usage may be meaningful, however, in the context of generic routines to avoid multiple writing down of similar branches. Then, we may enclose the generic in a kind of pseudo-procedure (generic, too), where the copying is done, and the generic itself may deal only with one structure over a variable base type, especially for library routines one must consider the trade-off between the copy overhead on the one hand and the possibility to use one name for one operation independently of types on the other hand.

Providing more type variability in PASCAL-GVT would have violated the above-mentioned principles of our extension.

The feature of the generic routine fits much better to PASCAL than that of type variability, since we have to exclude the grouping of types by predicates to retain static type checking. This feature

...
10th May, 1977

Dear Andy,

All is forgiven. Let’s forget the past and get on with work.

Distribution Centre for Australasia: I await your suggestions. I think we can act for Australia, New Zealand and Papua/New Guinea, but Japan is probably nearer the U.S. than us.

Standards: great news! desperately needed! I am on tenterhooks!

CDC-bias, etc: May I take some time to talk around the points you make on files, program, and CDC-bias? I don’t expect a reply since you are busy, but I’d like to try to convince you that I have some points here.

Your point (1): files as sequential access structures: I totally agree. Sequential files are useful; they are data objects; they are needed for the purpose you cite. What I try to say is that files are not full PASCAL-variables in that their usage in array of file or record of file, or file of file or in expressions, is undefined. They partake of few properties of full variables; about as many as procedure or function-names for example. I express a sadness that the opportunity has been lost of expressing this well in the PASCAL language.

Your point (2): arrays as random-access. I only partly agree. Sure a slow array could be implemented as a random-access file, but not all random-access files can be implemented as slow arrays. Unless you are willing to throw away PASCAL’s strong typing and admit truly dynamic sized arrays, the point being that even a random-access file is a sequence of variable length. PASCAL arrays are always of fixed pre-determined length. I emphasize that random-access is a property of the access, not of the file (though CDC’s standard implementation of files disguises this). Think abstractly. So I’ve no objection to slow arrays; they’re just not equivalent to random-access files.

Your point (3): program heading. I can’t see FORTRAN’s identical program heading as a “coincidence”, I’m afraid. Your subsequent argument is a pragmatic one for collecting all machine-dependent information at a central place. A good practical point. The counter-argument is that based on a feeling for structuring.

Besides this, several nasties creep in if the information is collected in the program heading. CDC PASCAL crudely restricts ‘permanent’ files to ones declared in the outermost block; if this restriction is lifted (an obvious step leading to better structuring of subprograms and their scopes), then name confusion may arise (two files called INPUT). In addition, the program heading could become very large for complex programs, and a useful facility has been pre-empted (I mean the facility to activate a PASCAL program with genuine parameters). How much of this is non-standard, but I hope it better illuminates what I mean when I say there is subtle CDC influence (and I mean subtle, not blatant: Wirth is a good designer).

It may interest you to have too that it is quite possible to leave attribute information out of a Burroughs B6700 PASCAL program, and to supply it all in the Back-Flows language (control program JCL) to be bound in at opening time. This puts machine- and non-dependent information right out of the PASCAL program, but would be unacceptably tedious if done for every PASCAL file or run. This would be self-evident to any B6700 programmer reading our documentation, but mightn’t be to others.

Your point (4): Remember B6700s and B5500s have been around a fair while; they are also today’s machines. My point here is that the CDC 6600s are just about an extreme in simplicity of (i) architecture, and (ii) operating system. It is quite natural that troubles will arise at the other ends of the spectrum. It is also quite natural that systems with reasonable affinity will prove to be easy to implement PASCAL on, for the assumptions are the same. Actually the CDC conventions and operating systems are more troublesome than its architecture which is a triumph of monolithic simplicity. Examples of the (again subtle) effects are PASCAL’s nonexistent attitude to interactivity, the lack of read/write scalars, and so on. Quite a long list of regrettable influences could be compiled. Many of them do not directly lead to implementation difficulty, but show up as a less-than-perfect construct. I grieve, but can do nothing about it.

Parenthetically, over the last 10 years I have had quite intimate contact with all the following systems: DEC, IBM, CDC, PDP-8, PDP-11 (20, 40, 65 870), Burroughs B7000, B8000, UNIVAC 1108, PDP-10, DECsystem 10 (KDF9), VAX and Interdata. I could add in more pre-1962 machines. I think I have managed to develop a connoisseur’s nose for machines and their deficiencies.

For a pervasive segment...

I am interested to know that the non-academic world in the U.S. is interested in PASCAL. I’d love to know how many of those PDP subscribers are (i) mini-computer firms, (ii) mainframe operators, (iii) software houses, or (iv) just interested individuals. It’d be interesting, yes? Thank you too for the Minnesota groundbreaking. 5 - 10s usage rate in number of runs is indeed good growth.

Our first-year course will switch over entirely to PASCAL next academic year (a first for reactionary Australia) now our compiler is operational, and I will put on a “What’s in PASCAL for you” course later this year for the general academic population. It will be interesting, as we are now preparing manuals and expectations have never been dominated due to some complex historical constraints. Switching Algol into Pascallers is easier in one way, but convincing them of the merit of the switch is more difficult!

We are also organizing through Burroughs to run our compiler on a B7000 system, and probably a dual-processor B6700. If I can get to any others of the range (eg the new 6800) I’ll try them too. We aim to thrash it on re-entrancy and any possible model-dependent features. Hardware documentation is very poor in Burroughs. And needed.

My best wishes. I hope the workload doesn’t get you down.

Yours sincerely,

[Signature]

10th May, 1977

The University of Tasmania
Postal Address: Box 252C, G.P.O., Hobart, Tasmania 7001, Australia
Telephone: 23 0501, Cables ‘Tasvec’ Telex: 58150

Open Forum for Members
Our other under-the-table extensions (type-functions which relax type-checking (cf. Richmond's transfer functions in PUGN #8) and the address-of-operator) illustrate more closely our ideas on why we feel no regret at "extending" Pascal. These systems-oriented changes were made for purely selfish reasons: some of us wanted to carry out systems programming entirely in Pascal, despite the fact that Pascal was not designed as such. The point is that programming in "extended Pascal" is much more satisfying than programming in an assembly language. Our concern is therefore that we should make Pascal more useful than it really is, simply because the alternatives available (on the CDC Cyber) are so abhorrent. In our minds, we always maintain the distinction between "Sydney" and "Standard" Pascal, and so does the compiler - it will, unless directed otherwise, flag every use of a Sydney-implemented extension.

Surely then, our efforts should not be concentrated on standardizing Pascal at a time when Pascal is beginning to show signs of age. There are non-trivial deficiencies in Pascal which are being attacked in more recent languages (Euclid, GL, Alfhard et al). Pascal might better serve therefore as a textbook in which improved ideas may be evaluated. I have this recurring nightmare: I'm reading the UTOPIA 84 Newsletter and they're complaining about all these old-fashioned people in industry and academia who won't move from Pascal to UTOPIA 84 because of the large financial investment tied up in Pascal software ... Pascal's role is not, I believe, to serve as the next important widely-used general-purpose language. It is a credit to its design that although it wasn't designed as such, it has nearly become such. Let's keep Pascal in its proper perspective, please!

Finally, we would be grateful if you would give our modifications some publicity. They are actually implemented, they work, our experience with them (over a year) is positive, and the implementation overhead incurred is definitely acceptable.

Keep printing,

(Tony Gerber)

(* Editor's Note: In a reply dated 77/06/07, I stated:

"I just received you letter, Tony, yesterday. John and I owe you several big apologies. I found out shortly after reading your 24 May letter that there was material on John's desk which I had never seen: a listing and some correspondence. I hope you don't get the idea that we go out of our way to hassle Australian PUG members! ...

"The trouble with an else on case is that it catches things you don't plan for as well as the things you do, and you can't distinguish among them. Separate compilation is a good thing. Your include feature or something like it will wind up in Release 3 [of Pascal-6000]."

"Regarding Utopia 84, I've had the same thoughts, but we haven't even gotten rid of Fortran yet, and once that precedent is set, getting rid of Pascal when its time comes will be easier. No, I don't think you comprehend the politics of getting a language like Pascal widespread. So yes, Pascal's role is to be the next widely used general purpose language, and any attempts by you or I are going to fail; it simply has too much merit on its own to stop it. Languages like Euclid, Alfhard, and GL are not general purpose and therein lies the rub! Besides they needlessly adopted different syntax for similar semantic constructs.

"Thank you again for all you have done..."

"P.S. What does "printing" mean?" *)
Dear Andy,

Mr. Andy Mickel
Pascal User's Group

June 1, 1977

Each Newsletter seems to be getting better. Number 8 is truly high quality both in presentation and content.

I have given lots of thought to the question of PASCAL software tools. There is no question that there exists a great need for the collection, review, and distribution of shareable software. We need to do this within PUG so that we can preserve our independence while increasing our scope.

Up until now I have collected and installed at Lehigh University a number of useful programs. I've used those to trade to get others. The problems of wider distribution have me truly worried. At Lehigh our antiquated 7 track drives and strange 63 character set make machine compatibility problems (via magnetic tape) almost insurmountable. I've even had five crates of cards (50,000) punched to import some software. Postage and other distribution costs have been paid out of my own pocket. There has got to be a better way - here's my suggestion:

I recommend that PUG Newsletter allocate a number of pages in each issue to the publication in source form of generally useful PASCAL programs. Both software tools and pedagogic examples could be published (program listings, documentation, designer commentary, and reviews) in "The Programmer's Corner" of the Newsletter. I could use my facilities at ANPA to produce camera ready copy of this material. Local non-standard usage could be clarified in text descriptions. Constructive criticism from members would be invited.

"The Programmer's Corner" has other benefits besides facilitating the sharing of programs. Good technique and compliance to standards would be encouraged. A new outlet for programmer/user ideas would be opened. Software tool distribution would be furthered by encouraging implementers/distributers to include the published programs on PASCAL distribution tapes. The tools would also form a good test base for implementors.

My personal interest in this stems from my great disappointment in the dropping of the Algorithms Section from Communications of the ACM. "The Programmer's Corner" offers a way to restore program and algorithm design to its rightful preeminent place in our profession.

I can see two disadvantages to this approach. First, it will take time before a thorough set of tools is published and, secondly, valuable space in the already crowded Newsletter will be used. To the first objection I respond that a continually growing, universally available software set offers significant advantages. To the second I offer the following method for increasing the available space in the Newsletter. First, we set up an abbreviation scheme. E.g., SA = slow arrays, DA = dynamic arrays, DF = direct access files, FIO = formatted input/output, etc. Letters from dissidents could then be tightly compressed for publication. "W/O FIO & DF, PU & WHRF: MH/ABT" could be the concise representation of "Dear Andy, Without formatted I/O and direct access files, PASCAL is useless and will never replace FORTRAN. ..."

Incidentally my own experience over the last five years with students who have learned to program using PASCAL is that if they go into a non-PASCAL environment, they quickly become an importer or implementer of PASCAL. In their minds, neither FORTRAN, COBOL, nor PL/I will ever replace PASCAL.

One final word about "The Programmer's Corner" idea. It seems to me that as our organization matures member interest will shift from implementation discussions to applications. I, therefore, look for the Newsletter to soon begin reflecting this change and membership to grow even faster because of it.

Sincerely,

Richard J. Cichelli
Research Manager
Computer Applications Department
Lehigh University

(* Editor's Note: I reacted negatively to this proposal at first, especially because of space considerations and who would judge what programs would be published. But Rich phoned me and talked me into it - provided he edit the section; he's really right that we should involve the interest of users much more than we have. It's been mostly implementors so far. Beginning with next issue (Pascal News #11), we should have some programs (mostly software tools) to print. See also the second page of Mike Ball's letter for his views on portable program exchange. *)

PASCAL NEWS #9 & #10
SEPTEMBER, 1977
PAGE 49
Mr Andy Mickel  
University Computer Center  
University of Minnesota  
Minneapolis, MN 55455

Dear Andy,

Enclosed is a check for my membership renewal for the next year. Please change my address to:

Michael S. Ball  
Code 632  
Naval Ocean Systems Center  
San Diego, CA 92152

This is a change of address due to a local reorganization.

I am currently hard at work on the concurrent and sequential Pascal compilers for the Interdata 8/32. The past few months were spent on the design of the Kernel and compiler changes, so I had very little time to worry about anything else. We have an initial operation date of 15 July, so things are coming to a head. It will not be available for distribution for at least several months.

The Univac 1100 compiler is seeing increasing local use, and there are 24 known copies in the field. There are 11 at universities, 4 at government installations, and the rest in industry. I have no data on the amount of use except at a few of the installations.

I was interested in the "standard extensions" to Pascal. I would like to suggest that these be limited to those which can be translated easily into equivalent standard Pascal. For instance, Dynamic arrays can be used in ways which are much more difficult to translate than can parameter arrays. Other extensions should be limited to additional standard procedures, and perhaps minor changes to highly system dependent actions such as file declarations. This limitation should increase program portability, while at the same time providing the convenience and added efficiency which seems to be the motivation behind most of the suggested improvements.

Along that line, I would like to suggest that a standard syntax be specified for external and other language subroutine declarations. The implementation is of course highly machine dependent, but a uniform syntax would ease transfer pains.

While on the subject of extensions, I heard from Jim Shores that you have a proposed extension for initialization which Wirth liked. If this is in shape for use, I would like a copy of this, since the initialization of tables is an area of considerable inefficiency in many programs.

I would also like to urge the creation of a standard editing procedure and distribution format for Pascal programs, since in my experience much of the trouble in transporting programs comes in incorporating corrections, and then later in merging corrections with the inevitable local modifications. Something similar to Bell lab's source code control system might provide a reasonable approach. The first job, of course, is to decide what features are needed, and what can be implemented in a portable manner. I would like to suggest the following list of features as a starting point.

1. The standard should include the full ASCII character set, but all programs should be case independent, so that they can be translated to an upper-case subset without harm.

2. Card length restrictions should be followed, since many operating systems work in card images. Serial numbers should be optional.

3. Corrections should include enough redundancy (perhaps an alphabetic checksum of some sort) so that corrections which are transmitted on paper have a reasonable chance of surviving the keypunching experience.

4. The system should provide the ability to add local changes with the local editor, then merge these corrections with new corrections from the distributor (a down-date procedure).

5. The programs which implement this should be implementable with the subset Pascals which are frequently the first step in a bootstrap. In particular, as few files as possible should be used.

More specific suggestions are easy to generate.

We are intending to implement some form of source code control system for our own use, and if there is interest in this, we will take the extra trouble to make it portable and generally useful. Let's hear from others on the subject. I am sure that I am not the only one tired of simulating other systems' editors by hand.

yours,

Mike Ball
incorporating Sir George Williams University and Loyola of Montreal

June 16, 1977

PASCAL User’s Group
C/o Andy Mickel
UCC:227 Exp. Engr.
University of Minnesota
Minneapolis, MN 55455

Dear Andy,

The merit of PASCAL is its simplicity. It is reasonable to expect a competent PASCAL programmer to correctly predict the effect of any well-constructed PASCAL statement, which is more than can be said of certain other programming languages. In attempting to standardize PASCAL we should attempt to tidy up loose ends, not to incorporate fancy features. When we have to extend the language, we should preserve the spirit of the initial design.

Everyone has their own ideas about what the most important defects of PASCAL are. My own pet grievance is the READ statement used to perform automatic conversion from character string to INTEGER or REAL. No user will accept a program which collapses when it encounters an unexpected character in the input stream, and no programmer wants to incorporate conversion procedures into every program he writes. Therefore, READ must have an error exit, and the program would not get a second chance to read the item. The solution should be compatible with the existing READ, so that simple-minded conversion is available for toy programs and novice programmers.

I tentatively propose the following: the READ statement should accept an actual parameter whose type is RECORD. The record must contain two fields, one scalar or REAL, and the other BOOLEAN. For example:

```
VAR ITEM: RECORD
  DATUM: SCALARTYPE;
  FOUND: BOOLEAN
END;
```

After executing READ(FILENAME,ITEM) either ITEM.FOUND = TRUE and ITEM.DATUM has the appropriate value or ITEM.FOUND = FALSE and ITEM.DATUM is undefined. In the first case, the file pointer will have been advanced past the item read, and in the second the file pointer will be unchanged, except that leading blanks and blank lines will have been skipped. If we have formatted input, then the pointer would be advanced over the indicated field width in either case, and the program would not get a second chance to read the item. If SCALARTYPE is INTEGER and the input stream contains

```
12345
```

then the first READ would find 123 and the next would fail, which might be confusing. We could insist that the item be followed by a blank, but this has obvious problems too. For example, a program reading expressions would accept 123 +456 but not 123+ 456.

The method extends naturally to user defined scalars and (not!) subrange subranges. This is important, because I think that it would be pointless to extend PASCAL in such a way that scalars could be read but entering FALSE instead of FALSE causes a fatal run-time error.

The programmer still has to provide an error recovery routine. For an interactive program, there is no problem: issue a diagnostic and cue for corrected data. For a batch program the easy way out is to READLN, leaving the user to spot further errors on the same line. In a specific application, however, it is often possible to design a more sophisticated error recovery procedure which takes reasonably intelligent action.

The PASCAL Newsletter is doing a fine job. Keep it up!

Sincerely,

Peter Gregor

---

June 24, 1977

Dear Andy:

Since it’s renewal time, I thought it would be appropriate to bring you up to date on PASCAL related happenings here at UT.

The best news is that we finally got confirmation that the new version of DEC-10 PASCAL has in fact made its way to the U.S. and DECUS. This confirmation came in the form of a copy of the files for a test installation from Carl Perkins of DEC to whom we had supplied the old version of PASCAL. He informed us that he would be the official DECUS submittor. We have the new version up and in reasonable shape. The biggest problem with it is that all programs that ran with the old version have to be changed.

On the Control Data side of things, Wilhelm Burger has left UT to take a job in Washington, D.C. Tom Koel of our staff is now looking after the PASCAL system. We are looking at installing your efficiency mods from the PASCAL Newsletter #5. Another programmer made a good start on a PASCAL interactive debugger this past semester.
Let me turn now to the question of standardization which has been debated so thoroughly in the PN issues of the past year. It appears from the information in PN #8 that the U.S. standardization process is not well understood. I enclose a copy of a presentation made at VIM-23 by MeredithSpeers which describes the process quite well. A careful review of the process will reveal that it is an extremely expensive and time consuming process. The effort in shepherding the proposal for a standard through SPARC is considerable. I would estimate that it would take a year and about $35,000 counting personnel support to get a technical committee set up. A conscientious effort could shorten this time frame, but I doubt it. Once the technical committee is established I suspect at least 12 to 18 months will be required to formulate an acceptable standard. Assuming quarterly meetings, this translates to 4 to 6 meetings. This estimate assumes a 20 to 25 person technical committee. As you point out in PN #8, the technical committee is critical to the formulation of a standard and I doubt that the canvas approach will work with PASCAL given the acknowledged weak spots in both the Report and Manual.

The technical committee under X3 rules is a volunteer organization with strong continuing attendance requirements to assure a body of expertise behind the proposed standard. Given the strong interest in a standard expressed within PUG, I would expect a technical committee of 20 to 25 sufficiently committed volunteers could produce a standard in 12 to 18 months. The most difficult part, as you point out, would be to control extensions to the language.

If the effort through BSI does in fact result in a proposed ISO standard, then SPARC will almost certainly set up an X3 PASCAL technical committee. Consequently, I think that a U.S. X33 committee for PASCAL is probably inevitable and PUG should probably take the leadership in establishing such a committee.

Enclosed is my renewal check. Keep up the good work!

Sincerely,

Waldo M. Wedel, Manager
Programming Services

Mr. Andy Mickel
University Computer Center
227 Exp. Engineering Building
University of Minnesota
Minneapolis, MN 55455

Dear Andy:

Please find enclosed my membership for the next academic year for the Pascal User's Group. And congratulations to you, John, and the others for producing four newsletters of exceptionally high quality. Keep up the good work.

After reading Newsletter #8 and listening to CDC present their future plans, I agree with your position that now is the time to formalize the definition of Standard Pascal by cleaning up the semantic definition and making relatively few extensions to the syntax. The important syntactical changes should include dynamic arrays, value initialization (including arrays and records), strict procedure parameter type checking and case statement alternative.

I don't expect to see the bulk of my proposals in Newsletter #8 implemented in Standard Pascal. I believe the best route for implementing extensions to PASCAL is to build a preprocessor (written in Standard Pascal) to translate extended Pascal to Standard Pascal. Such a processor is truly portable and essentially changes the compiler into a two-pass system.

Our distribution mechanism is operating efficiently with less than one week turnaround (except for vacations). Karin Bruce and Michele Dowd are doing a good job. I've enclosed some of our recently developed material. Karin feels it would be more expedient to drop the option of letting the buyer supply the tape and incorporate the cost of a tape into the minimum cost. I concur with this idea. Do you have an opinion on this change?

*****

Sincerely,

George H. Richmond
July 28, 1977

Mr. Andy Michel
Editor, Pascal User's Group
University Computer Center
227 Experimental Engineering Bldg.
Minneapolis, Minnesota 55455

Dear Andy:

I thought your readers might like to know that we have an interesting PASCAL project in progress and that there are PASCAL related positions available here in Ann Arbor.

ADP Network Services currently operates more than fifteen DEC System-10's on an international communications network and we have the need to develop a systems implementation language to support language, monitor and other software development for DEC-10's and other hardware that may be attached to our network as our company grows. PASCAL has been chosen as the base for this language. We have embarked on a joint project with Al Kortesoja of Manufacturing Data Systems Inc., also of Ann Arbor, to develop language and code generation features that will provide us with a general implementation language that will generate code for a variety of machines.

We began with the DECsystem-10 compiler developed by H. Nagel of the University of Hamburg and are modifying it to include: random I/O facilities; flexible length arrays; constant arrays and records; an exception handling facility; functions which return arrays, records and sets; and STRING handling. Through this we have endeavored to maintain the coherence and compile time checking capabilities originally designed into PASCAL by Professor Wirth.

We, ADP Network Services and MDSI, have a variety of positions open in the areas of PASCAL compiler development, systems programming, and applications development using our PASCAL. I would be pleased to receive any resumes your readers would like to send and would see that they are properly considered by Mr. Kortesoja and myself.

Sincerely,

Neil J. Karta
Manager, Programming Languages

network services, Inc.
175 Jackson Plaza
Ann Arbor, Michigan 48106
(313) 768-6800

Dear Andy,

Enclosed is my renewal; if I've missed P.N. 89, would you send me a copy?

I really stand in awe of the job you've done in publishing the P.N.; nevertheless, I hereby add to your burden with the following.

If PASCAL is to compete with Fortran, I believe four things are needed which I have not seen discussed as a unit in the Pascal Newsletter so far; hence, this letter.

Before I go on, I should point out that all the possibilities discussed here can be inserted into the Pascal language without much syntactic change. Better still, efficient one-pass compilation of these features is still possible, Fortran being a weak demonstration of the fact, another being found in an M.Sc. thesis which discusses these and many other interesting possibilities, "Pyxis: A Language Evolved from Pascal" by E. N. Kittlitz, Department Computer Science, University of Calgary, 1977. (The author may be contacted via that department, Calgary, Alberta, Canada T2N 1N4.)

First, concerning storage mapping: I join the cry for a variable initialization facility, which in turn implies a certain amount of statically allocated storage.

Second, storage could be explicitly allocated as static either in common blocks or as "private" areas for given procedures or functions. Then one has the possibility of Pascal subroutines that do not use the run-time stack and so could be loaded with and called by a Fortran main program. A second benefit that I find personally more important is that one could then program more modularly, no longer having to use unprotected globals to implement the Algol own.

Third, there is the need for flexible array parameters; I don't suppose that is debatable any more. Of course, one must distinguish between flexible array parameters and "rubber" dynamically-allocated arrays. It strikes me as not in the spirit of Pascal to admit rubber arrays, nor would rubber arrays be at all necessary from the view of Pascal as a Fortran-replacement.

The flex array facility of Pyxis has merit; for example, it costs nothing if you don't use it. The following is a Pyxis program fragment which prints the sums of the two 6-element vectors.

```pascal
type Flexvec = array [1 to *] of real;
var A: array [5 to 10] of real;
B: array [-3 to 2] of real;
function Sumvec(X: Flexvec): real;
var I: integer;
S: real;
begin S := 0;
  for I := 1 to UPB(X) do S := S+X[I];
return S;
end;
begin (*initialize values*);
  write(Sumvec(A), Sumvec(B));
end;
```
Pyxis also allows one to allocate flex-typed objects of run-time-specified size to the heap, and to have a pointer which may reference any object of a given flex type, i.e. an object of a type which falls within the class of types specified by a flex type declaration.

The fourth point involves the great format debate, and variant records too. I think people are not thinking straight about these issues. A text file is not a string, nor a sequence - not even one of indeterminate length! It has funny states, e.g. the "not-opened" state; even an abstract model of a file does odd things. In Pyxis, a program interacts with a file (which is "outside" the program) via its image (which is a record of status information with a string acting as a buffer); a string is a fixed-length packed array of characters, in the Pascal sense. Thus, format operations become type coercions changing various simple data types into short strings and vice-versa; the analogy with integer-to-real coercion is quite good, and format operations are no longer the perquisite of the file handling package.

Of course, not all the foregoing viewpoint fits well with Pascal, but some fair amount does, and is worth considering. Assuming a good type coercion syntax can be designed, format operations could simply be functions which accept or return flexible arrays of characters, and their use in I/O becomes natural without being their only use. Further, if you do not use these functions, they need not consume space in your load module.

The tie-ins with variant records should be clear. Variant records are used for two totally distinct and completely valid reasons. The first is that which they were designed for; the second is to pun: One must write one's own "dispose"; one needs to dump large list structures; and for a myriad of other purposes a programmer sometimes needs to get at the bits, do arithmetic on pointers and the like. Although these activities are machine dependent they are not dirty; because they must be done with great care, they must be done in a good language; and because they are so universally necessary, they ought to be accommodated in the language in a clear machine-independent way.

Rather than continuing the abuse of the variant record, let the job be done by a syntax designed for the purpose. To this end, I favor the common idea of allowing a <type identifier> to be used as a <function name> such that if its (one) argument is of a suitable type, a pun is allowed or, in certain specified cases, a coercion occurs. A suitable type for punning would normally be a type requiring the same storage as that which one is "pulling it into"; and if the user doesn't know his implementation well enough to do what is required, he's still better off with the resulting error message than with the current "guess and hope" method required by variant records.

In summary, I hope most for variable initialization, private (own) variables, flexible array parameters (but not rubber arrays), and a view of type coercion to solve both formatting and variant-record problems.

Killing Fortran was presented as a motivation; more precisely I want a strong, viable language so I won't have to reprogram soon. I've done a lot of work in Pascal, in part because I hope that with just a little more strength of expression Pascal will survive; but I also believe that without that strength, it won't.

Very truly yours,

Stephen Soule,
Assistant Professor

SPECIAL TOPIC: MICRO/PERSOMAL COMPUTERS AND PASCAL

The following four letters deal with some developments described on page two of the EDITOR'S CONTRIBUTION. See also the IMPLEMENTATION NOTES section under INTEL 8080, LSI-11 Motorola 6800, etc. And also see HERE AND THERE News section under Kenneth Bowles, Kurt Cockrum, John Collins, Creative Computing, Jack Crone, Dan Fylstra, Roger Gulbranson, Carl Helmers, Sam Mills, Aron K. Insinga, Barbara I. Karkutt, Ed Keith, Donald Lindsay, Tim L. Lowery, Bruce Mackenzie, Jim McCord, Carlton Mills, Carol Anne Ogdon, David Segal, Bruce Seller, Michael Settle, Jeffrey G. Shaw, David H. Welch, and Richard West!

Andy Mickel, Editor
Pascal Newsletter
University Computer Center
227 Exp Engr
University of Minnesota
Minneapolis, MN 55455

Dear Mr. Mickel:

(1) I have received a reply from Dean Brown at Zilog about the hypothetical Pascal machine. Zilog is not describing the machine to the public at this time, hence enclosed copy. Perhaps his spontaneous use of the term "Pascal machine" is a hopeful indication however.

(2) Enclosed please find copies of letters I have sent to Byte, Creative Computing, Kilobaud and Personal Computing as my one-man campaign to stamp out BASIC and increase Pascal's visibility.

(3) Since (judging from PN 8) Pascal will soon be available for personal computers, it seems to me that a timely collection of Pascal games and hobby programs might be of great interest. I personally have been writing Pascal versions of Star Trek, MasterMind, Lunar Lander and so on. I would like to hear from anyone in PUG interested in sharing such programs, and also from anyone who could explain to me the copyright laws concerning Pascal translations of copyrighted BASIC programs.

(4) I personally was aghast at the proposal to change variant record usage (PN 8-15). I think the language designer's responsibility to protect the programmer from himself stops short of that. Perhaps I have strange tastes, but I like having access to individual bits of a word by treating the word as a packed array of boolean. I like being able to declare

```pascal
var r: record case boolean of false: (x,y,z:integer);
true: (parray[l..3] of integer) end;
```

so that for statements can be used for assignment (for i := 1 to 3 do p[i]:=something) yet clumsy array notation is avoided in other situations, for example:

```pascal
write(a[x,y,z]) instead of write(a[p[1], p[2], p[3]]).
```

(5) Could someone in PUG explain why Pascal's semicolons make Prof. Sales weep? (PN 8-33)

(6) Congratulations on the Newsletter.

Sincerely,

David A. Mandle

8 July 1977
June 27, 1977

Dear Andy,

Thank you for the copy of your newsletter. I will put a "short contribution" extolling it in the next issue of Interface.

As Steve Legenhausen points out on page two of the newsletter, BASIC is becoming a microcomputer standard. I am very much interested in urging our members to consider other languages than BASIC, and would like to publish anything which would work to that end. An article such as a Pascal tutorial, a critique of BASIC (control structures, data types, etc.), a Pascal bibliography, a survey of micro-based Pascal activity, or a Pascal subset proposal, would be most valuable for our readers.

If you or any PUG members would be up for writing or compiling material along these lines, I would love to publish it. Like yours, our format is quite flexible, with room for short contributions as well as longer articles.

Sincerely,

Larry Press
Editor

P.S. We have an informal system of coordinators for various topics. Would you mind if I were to list you or PUG as coordinator for Pascal?
Dear Andy,

I've received issues 5-8 of the PUG Newsletter, and am mightily impressed with the sheer volume of (largely useful and interesting) material you have managed to compile and publish.

In reference to my earlier offer to help promote PASCAL, you mentioned "pressing our advantage in the microprocessor area" through articles and letters to such magazines and journals as Dr. Dobbs' Jolt, Personal Computing, Creative Computing, etc. While I'd be glad to help these and other publications with pre-PASCAL material, I really can't "press" any "advantage" because, frankly, we have none... yet. As of today, I know of no reasonably-priced, memory-efficient, generally available implementation of PASCAL (or at least usable) in compiler or interpreter form, suitable for use on any of the popular micros, with the dubious exception of the LSI-11, which has itself only become inexpensively available through the still-brand new Heath computer line.

Having an "advantage" entails, for me, two considerations. First, one's product or service must be inherently superior to its competition. Secondly, it must be available and easy to use. PASCAL certainly is a superior language, perhaps the best I've yet encountered, despite its many flaws which I hope will be truly CORRECTED, and not merely "written around". However, the availability of a powerful, easy-to-use micro-PASCAL remains nil, and so our "advantage" remains merely a tantalizing phantom. For the average micro-user, PASCAL is, and will remain, "unreal" until someone comes up with an implementation which is, from both aesthetic and practical standpoints, more attractive than the alternatives BASIC and FORTRAN. (Micro-PASCAL will have to be "more attractive", of course, in order to lure away the vast majority of satisfied BASIC and FORTRAN users, and give them proper cause to learn and embrace a strange new language.)

I've been reading about the UCSD PASCAL project, and I'm filled with hope that, finally, I will be able to show my doubting friends and customers something more than the [often confusing] User Manual and Report. Perhaps I will be able to demonstrate a working compiler or interpreter, as well as the superiority of PASCAL as a programming tool. The moral victory would be even greater if I could point to simultaneously compilable versions of the language optimised for the LSI 11, 280, 8080 and 68021. Anyway, until I hear more from La Jolla, the emergence of PASCAL into the micro-age is still in my pipe dream.

Regarding media exposure for PASCAL, though, I am all for it, and suggest the formation of a steering or co-ordination committee to manage a media blitz to awaken the personal computing community to the advantages and joys of PASCAL programming. What do you think? I have noticed too many APL articles popping up lately, and suspect that either co-incidence is working against us, or an APL blitz committee (formal or informal) has been formed and is calling the shots. In either case, we'd better get something together if we intend to make any dent in the personal computing sector. APL, as cryptic as it is, is still a good language, and could very well bury us by default if we don't watch out.

Finally, in PUG #8 [I think], you expressed interest in getting informa-
Dear Andy,

Finally getting around to a detailed reading of PUG Newsletter #8 provides me with a theme for an editorial I will put into the December 1977, pushing PASCAL as a possible language. I picked up several Springer-Verlag books at IFIPS last month and have since spent some time discussing PASCAL with my good friend and associate Dan Fylstra.

I think that PASCAL would make an excellent choice as a successor to BASIC in the personal computing world, a thought which is echoed by several contributors to PUGN #8. Here are two points about PASCAL Personal Computing Use which will no doubt appear in the editorial I am composing this week:

* Like BASIC, PASCAL is an academically originated language with a fairly well defined set of machine independent standards. As such it has one major point in its favor: it is not a proprietary product confined to any one organization, and is thus open to the general computing public as a standard to be implemented and delivered with machines. Thinking of the general public as users requires a machine independent (or nearly so) language, and in the interests of better software techniques a structured language like PASCAL comes to mind. The large amount of activity evidenced by PUGN suggests that both the academic training and wide usage which were present in BASIC's evolution will also be available with PASCAL.

* When implemented for the personal computing milieu, PASCAL should at a minimum level of function offer an interpretive or semi-compiled interactive system which is friendly to the user in the same way that BASIC is friendly. Fully compiled and optimized code generation is not needed in a context where one high speed processor is dedicated to each user and his or her files.

As a final point in closing, we (BYTE Publications) are in the process of preparing a series of publications initially oriented to systems software books characterized by tutorial documentation of the design, complete publication of source code and any necessary intermediates, machine readable representations of the source of object text, and other information relevant to the process of getting the particular software item running in the user's personal system. (Where machine dependence is involved, we are looking for target machines which are in the following set: LSI-11, 6800, 6502, 8080, Z-80.)

I would be most interested in talking with readers of PUGN who have implementations of PASCAL available for sale which run interpretively, semi-interpreted, or as compilers. Our standard form of publication agreement is an exclusive book and audio record publishing license to the software and its machine readable representations.

I'll send a copy of the editorial after it is written.

Sincerely,

Carl T. Helmers
Editor in Chief
BYTE Publications, Inc.
Among the actions described as being taken were: 1) we try to clarify instances of vague semantics in the Revised Report, and 2) Tony Addyman of the University of Manchester coordinate an effort to get an ISO standard certified which would amount to a "tightened up" Revised Report with no additions.

This summer, Tony phoned that: 1) he had received another list of points requiring clarification from Jim Welsh in Belfast. 2) he wondered if there would be copyright problems with the current Revised Report already published and the proposed standards document.

I sent him a small list of items which included:

1) Optional; on last limb of a case.
2) Role of the word-symbols: extern, forward, and fortran (all non-standard) but in example (brought to my attention by Charles Hedrick): the Report specifies the mod operator as the operation: "modulus." But the mathematical meaning of modulus gives things like: -3 mod 2 = -1. The Axiomatic Definition clearly states -3 mod 2 = 1.

On 77/08/17 I received a note from PUG member D. G. Burnett-Hall dated 77/08/10 which read: "Dear Andy,

I enclose 'Another Attention List' following Tony Addyman's Attention List in Newsletter 8: I've tried to avoid duplicating his points (and I've sent him a copy)."

Another Attention List

D. G. Burnett-Hall

1977 August 9

DEPARTMENT OF COMPUTER SCIENCE

Section

4

(a) Add "programs" to first sentence.
(b) Is " an illegal constant? (n = 0 characters not defined.)

6.1.1 type T1 = (ZERO,ONE); T2 = (ONE,TWO);
should be illegal because the type of ONE is ambiguous (UM-5A,p34)

6.1.2 For Boolean type, better to make clear here that it is ordered (false, true) than just a note in 8.1.4.

6.1.3 Allows lower upper band for subrange type: UM-5B(p35) does not. (Why?)

6.2.1 (a) Is array [integer] of real legal?? Note that <index type> ::= <simple type> , and <simple type> ::= ... [:] <type identifier>.
(b) type T1 = array [0..9] of array [boolean] of integer; T2 = array [0..9, boolean] of integer; var A1:T1; A2:T2;
Are T1 and T2 identical types? (Assuming that "identical types" means more than having the same type identifier.) Specifically, is it legitimate to write A1 [5,true] or A2[5][true]?
(cf. UM-6, p39)

6.2.2 (a) Field identifiers within a record must be distinct, taking all variants of the record into account (UM-7,p46).
(b) Helpful if last example included an empty field list (UM-7, p46).

8.1 This should include something along the lines of UM-4A(p21) and UM-10, p63/64) about whether compound Boolean expressions are completely evaluated. (*It would change the language to say now that they are evaluated only as far as necessary, but I wish this had been done. So did the author of SKILLPLANS (UM-12A,p65) which is obviously legal.*)

8.1.2 (a) 14 div(-3) is not defined anywhere! Is it -4 or -5?
(b) mod operator is defined (in terms of div) only in UM-2B(p3).


9.1.1 Consider also:

(a) case b : Boolean of false : (1:integer);
   true : (R:real)
   end;
   var X,Y,Z:R4;
   (*Integer and real quantities need not be the same size*)
   X.B := false; Y.B := false; Z.B := true;
   X.1 := 1; Y.1 := 2; Z.1 := 3.4;
   X := Y; (*Presumably their types are identical*)
   X := Z; (*Legal? Are their types identical?"
   (*Does this imply a run-time check?*)

9.1.2 UM-11A(p71) says that if procedure P (var X:integer);...
   is declared, the procedure statement
   P[A,A]
   is illegal ("xl..xn should be distinct variables"). Why?
9.1.3 (a) Doesn't forbid duplicate use of one label in the same block!
   (b) procedure P;
       begin (*P*); goto 99;
   procedure Q: begin ...... 99; ...... end (*Q*);
   R.9.1.3.1 does not require label 99 to be in Q, thus
   contradicting its second sentence. In R9.1.3.2, should
   "procedure" be replaced by "innermost procedure, function
   or program?"

9.2.2 (a) All the case labels within one case statement must be
   distinct (UM-402, p31).
   (b) One of the examples should include a list of case labels.
   (c) A case label cannot be used as the destination of a goto
   statement. (This is implicit, but it would be helpful to
   make it explicit.) In the final example, it is very
   tempting to write:
      1. begin x := x - pi/2; goto 2 end;
   9.2.3.3 (a) Never says that the statement S will not be obeyed
   if el>e2 (to) or el <e2 (downto)!
   (b) The semantic explanations
      (i) should be enclosed in begin ... end;
      (ii) firmly state that the final value of v is e2 (*if S
   does not cause a jump*). And why not? (Except
   that UM-4C2, p24, says it is "undefined".)

10 (a) forward is not mentioned at all. (cf. UM-1IC1, p82/83,
   which consistently does not treat it as a special symbol:
   surely this is a mistake!) Is this a case for a "con-
   ventionalised extension"?
   (b) If a procedure or function is used as a formal parameter,
   UM-1IC2 (p83) states that the corresponding actual
   procedure/function must take value parameters only.
   (*And so it can't itself have a procedure or function
   parameter: for this relief much thanks.*)

10.1 I should favour making HALT a standard procedure (for use after
   disastrous errors). (*It can be done by goto 9999, where 9999:
   is at the end of the program, but that may involve exit from
   procedures, and HALT is much simpler.*)

10.1.2 NEW(F,T1,....,TN) : the wording does not make clear whether the
tag-field values are assigned. UM-10(p64) emphasizes that
no assignment takes place.

11.1.3 I assumed that TRUNCX was the mathematical function [X]:
     UM-2B(p13) says TRUNC (-3.7) = -3, not -4. The present wording
implies sign-and-modulus representation of numbers.

11.1.4 PRED and SUCC: type of argument must exclude real.
     (UM-2C, p14)

12 Can one assume, when reading a textfile F, that eof(F) becomes
   true only (i) after readln (f) or (ii) immediately, on reset (f)?
   (i.e. the file ends with the end-of-line marker or is empty?)
If not, the program schemata in UM-9A2, 9A3 (p58/59) will fail.
This assumption would considerably simplify input of data (try
rewriting UM-9A2 if eof can occur at any moment!) and would be
easily implemented in the run-time system. Perhaps this should
be answered in R14?

12.1.4 (a) var Rrisal: I:integer; F:text;
     .... read (F,R,!) ....
     If the data are +2 -13.4,
     — is it legal to give integer datum for real variable
      (cf r :=2)?
     — is it illegal to give real datum for integer variable
      (cf i :=-13.4)?
     (*DEC-9 system rejects both: the first is unhelpful and
      unnecessary.*)
     (b) When read (F,V) finishes reading a number, is F! the
     character which terminated the number (and not the one
     following it)? UM-App.F suggests it is.
     (c) What terminates (i) an integer, (ii) a real number?
     Does end-of-file terminate an otherwise valid number?
     (*My suggestion under 12 would remove this possibility*)

12.3.5 If e is non-negative, should it be preceded by (i) 'e' or
     (ii) at least one space, or (iii) are no preceding spaces
     required? (UM-12/7, p86, suggests the last.) Should this be
     answered in R14?

12.3.6 (a) UM-12/7,128B (p86/87) is considerably more helpful, but
     still impricise.
   (b) What, if anything, replaces the + sign for (i) a positive
     number, and (ii) a positive exponent? (UM-App.F answers
     (i) one space, (ii) '+'.) Should this be answered in
     R14?

12.3.9 The last 3 lines of R12.2 should be moved to the end of R12.3.
     Should they be numbered 12.3.9? (cf last 3 lines of R12.2)

12.1.2, 12.1.3, 12.2.2, 12.2.3, 12.3.2, 12.3.4, 12.4.2:
enclose equivalent statements between begin ... end.

13 Why is reset (input) illegal, and not just of no effect?
   (And rewrite (output), for that matter?) If one can reset (F)
   for a textfile F to re-read its data, why not for INPUT?
   (*DEC-9 system allows these: chiefly because DEC-9 operating
   system expects matching of internal and external filenames to
   be done at run-time and not by JCL commands, worse luck!*)

14 (a) This section should include a list of those implementation-
   dependent constants whose values are needed when a program
   is being transported. e.g. MAXINT (UM-28, p13), but not in
   Report), maximum size of a set, default values of M in write
   (E:M) for integral and real E, etc. (*Should these be
   standard constant identifiers? I favour a limited number
   of environment enquiries, c.f. Algol 68*)
   (b) A compiler-option takes the form of a comment whose first
character is $. (That much, pace N. Wirth, but no more.
It ensures that one can put comments in portable programs
which won't accidentally be taken as compiler options.*)
Implementation Notes

CHECKLIST CHECKLIST CHECKLIST CHECKLIST

1. DISTRIBUTOR/IMPLEMENTOR/MAINTAINER
   (* Names, addresses, phone numbers. *)
2. MACHINE
   (* Manufacturer, model/series and equivalents. *)
3. SYSTEM CONFIGURATION
   (* Operating system, minimum hardware, etc. *)
4. DISTRIBUTION
   (* Cost, magnetic tape formats, etc. *)
5. DOCUMENTATION
6. MAINTENANCE POLICY
   (* How long? Accept bug reports? Future development plans. *)
7. STANDARD
   (* Implements full standard? Why not? What is different? *)
8. MEASUREMENTS
   (* - Compilation speed (in characters/sec. please; this is a meaningful measurement for compilation speed); - Compilation space (memory required at compilation time); - Execution speed; - Execution space (the memory required at execution time; compactness of object code produced by the compiler); ** Try to compare these measurements to the other language processors on the machine, e.g., FORTRAN. *)
9. RELIABILITY
   (* Stability of system (poor, moderate, good, excellent); how many sites are using it? when was the system first released to these sites? *)
10. DEVELOPMENT METHOD
    (* Compiler or interpreter? Developed from Pascal-P / hand-coded from scratch/bootstrapped/cross-Compiled/etc.? What language? Length in source lines? Effort to implement in person-months? Previous experience of implementors? *)
11. LIBRARY SUPPORT

GENERAL INFORMATION

As this is the first issue of Pascal News in this academic year July 1, 1977 - June 30, 1978, let us explain how this section is organized:

-- First a CHECKLIST to be used as a guide to distributors, implementors and maintainers for reporting the status of Pascal implementations on various computer systems.
-- A SOFTWARE TOOLS section describing aids to Pascal users in developing applications.
-- A PORTABLE PASCALs section reporting distribution information about kits used to produce Pascal compilers for real computer systems.
-- Information on PASCAL VARIANTS.
-- A FEATURE IMPLEMENTATION NOTES section describing implementation strategies and details of various Pascal features as suggestions to all the compiler implementation efforts underway.
-- A list of MACHINE DEPENDENT IMPLEMENTATIONS sorted by name of computer system, giving news of Pascal compilers for real machines.
-- And in subsequent issues this year, an INDEX to all the implementation information for this year.

We are essentially beginning anew this year and so in this issue we are combining summaries hand-compiled from PUGN's 5-8 (last year) with the news received since #8.

IMPLEMENTORS - MAINTAINERS - DISTRIBUTORS

Please use the checklist if you are reporting information and please keep us all up to date. You might also send us a copy of your documentation and distribution forms as so many implementors have done so that we can keep up to date on the overall development of Pascal. Please send camera-ready copy, single spaced, and use wide text (we prefer 18.5 cm lines). We also will accept reports on ASCII paper tape accompanied by a listing. And please include PUG All-Purpose Coupons with each copy of your system that you send out!

USERS

Please help make us all informed consumers of Standard Pascal systems by reporting your quantitative and qualitative experiences with particular implementations.

EVERYONE

We would like to thank all the effort put forth by people who have sent in information. We regret to say that our ability to answer individual requests is limited not only by time, but by the commitment we have first to this section of the newsletter. Therefore we prefer to answer inquiries through Pascal News. We print all the news that comes to our attention barring oversights and mistakes on our part!
SOFTWARE TOOLS

There has been much discussion concerning the distribution of Software Tools written in Pascal in PUGM 5-8. Please see the letter by Richard J. Cichelli in the OPEN FORUM section this issue. It was our idea that tools should be incorporated with the distribution of Pascal implementations but even this poses real problems. Starting next issue we should see news on tools greatly expand.

Examples of software tools are listed below:

1. A program to cross reference identifiers in Pascal programs.
2. A decompiler to examine the object code produced by a Pascal compiler.
3. A prettyprinter to format and indent Pascal programs.
4. Performance measurement programs to monitor execution times in Pascal programs.
5. A program to change character sets from one to another.
6. A program to compare two text files and generate a set of modifications to convert one to the other.
7. A text editor to alter and record modifications made to a source program.
8. A text formatter used to produce documentation about software for users.

Believe it or not, right now several different programs exist in categories 1, 4, and 8, and at least one program exists in every category except 7! All are written in Pascal.

PORTABLE PASCALS

Pascal-P.

The most widely used portable compiler for creating new Pascal implementations is Pascal-P. Basically Pascal-P is distributed from three points in the form of a kit consisting of a magnetic tape and printed documentation.

Pascal-P is a compiler written in Pascal (almost 4000 lines) which generates symbolic code for a hypothetical stack machine called a "P-machine" because it is somewhat of an ideal architecture for Pascal-P. The symbolic code is thus called P-code.

The person implementing Pascal has several choices. If there is no access to a working Pascal compiler on another machine, the implementor orders a Pascal-P kit already configured to the target machine. Configured compilers have constants inserted in them to specify, for example, the size of each simple data type. These configuration parameters are given by the implementor on the Pascal-P order form. (See below.)

After receiving the kit, the implementor can write an interpreter for P-code in another language (usually takes about one person-month), and thus immediately has access to a Pascal compiler running interpretively by using the P-code version of the compiler included in the kit.

To produce a real Pascal compiler for the target machine then requires editing of the Pascal-P compiler written in Pascal to produce code for the target machine (instead of the P-machine). After recompiling, a Pascal compiler exists in the code of the target machine.

If the implementor initially has access to a working Pascal compiler on another machine, the step of writing a P-code interpreter can be omitted.

Facts about the Pascal-P compiler:
- The current version is called Pascal-P4 and is distributed with a copy of Pascal-P3 (which is of interest to previous recipients of Pascal-P2).
- Pascal-P4 represents a major improvement over earlier Pascal-P versions because it removes data type alignment restrictions, is more efficient, includes runtime tests, and is a more complete implementation of Pascal.
- Pascal-P3 was developed from a phase in the stepwise refinement of Urs Ammann's Pascal-6000 compiler in 1974 by K. W. Norl, Urs Ammann, K. Jensen, and H. H. Nageli. Subsequent improvements were done by Christian Jacobi.
- Reliability of Pascal-P4 has been fairly good. As of Spring, 1977, it was distributed to 106 sites by George Richmond and to 37 sites by Chris Jacobi. (No distribution data has been received from Carroll Morgan.)
- Several good reports on the viability of Pascal-P were reported in PUGM #8 as well as two more in this issue: Ted Park for a Data General Nova and John C. Knight for a CDC Star=100.
- There is no promise of maintenance for Pascal-P. P4 is the final version produced at Zuerich. We at Pascal News will attempt to print bug corrections in future issues in this section.
- Documentation for Pascal-P4 consists of a 65 page report entitled The Pascal-P Compiler: Implementation Notes (Revised Edition) July, 1976. (A 24 page correction list to the original December, 1974, edition is also available.)
- Pascal-P4 is a significant subset of Standard Pascal. Restrictions to the standard include:
  - procedure and function formal parameters are not allowed.
  - files are not implemented.
  - goto's may not exit procedures or functions.
  - a (rather small) maximum string constant length is imposed.
  - the subrange form of set constants is not implemented.
  - nil is not a reserved word, but rather is predeclared.
  - many standard procedures and functions are not fully implemented.
  - text and maxint are not predeclared by the compiler.

Pascal-P can be ordered from three places (write for prices and order forms).

Implementation Notes
In North and South America, order from:  
George H. Richmond  
Computing Center: 3645 Marine Street  
University of Colorado  
Boulder, CO 80309  
USA  
Phone: 303/492-8131

In Australasia order from:  
Carroll Morgan  
Basser Dept. of Computer Science  
University of Sydney  
Sydney, NSW 2006  
Australia  
Phone: 629 1122

(*) We at PUGN would appreciate new ordering information be sent to us by these three distributors for inclusion in Pascal News #1. We would also appreciate some sort of coordination on a common order form for these three distributors.*

Pascal Trunk Compiler

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. H. H. Nageli, Institut fuer Informatik, ETH-Zentrum, CH-8092 Zurich, Switzerland (Tel. 32 62 11).
2. MACHINE. The trunk compiler is the machine independent part (e.g., syntax analysis and error recovery) of a Pascal compiler in which the code generation has to be inserted in a certain number of empty procedures.
3. SYSTEM CONFIGURATION. Requires a working Pascal compiler.
4. DISTRIBUTION. Magnetic tape. Cost: SFr. 50.--.
5. DOCUMENTATION. In German, available in May, 1977 (77/3/3). Detailed comments in the source describe how an implementor can write algorithms for the machine dependent parts.
6. MAINTENANCE. Not defined yet.
7. STANDARD. Full Pascal is treated.
8. MEASUREMENTS. Not applicable.
9. RELIABILITY. Moderate (77/3/3). The Trunk was used with good results in 1975-76 by Teruo Hikita in producing a high quality Pascal compiler for the Hitachi 8000 series.
10. DEVELOPMENT METHOD. Started in 1975 in a phase in the stepwise refinement of Urs Ammann's Pascal-6000 compiler. The Trunk is a 5800 (indented) line Pascal source program in which the machine dependent parts are clearly marked and separated from the machine independent parts.
11. LIBRARY SUPPORT. Not applicable.

Some other machine-dependent compilers are written in such a way that they might be useful as Trunk compilers. Take for example, the current ICL 1900 compiler written by Jim Welsh, Colum Quinn, and Kathleen McShane at the Computer Science Department, Queen's University, Belfast, Northern Ireland, BT7 1NR, United Kingdom. The syntax analysis is clearly separated from the code generation in this compiler, which is written in Pascal. See ICL 1900 under Machine Dependent Implementations.

Another possible Trunk-like compiler is that implemented by Alain Tisserant, Department Informatique de l'INPL, Ecole des Mines, Parc de Saurupt, F-54042 Nancy Cedex, France. In this case, the compiler operates in two passes; the first pass can be parameterized and the second pass can be rewritten to generate code for different machines. This effort is explicitly oriented toward 16-bit machines. As far as we know, no other implementations have been developed from the initial compiler. See SHE'S T1600 in the Machine Dependent Implementations section.

Pascal J

2. MACHINE. Pascal-J is a compiler which translates Pascal to the intermediate language Janus, a totally portable "mobile programming system" — even to the point of defining its own character set! Janus is in turn is macro-processed via Stage2 which is implemented in standard Fortran.
3. SYSTEM CONFIGURATION. ANSI Standard 1966 Fortran IV compiler. Specify character set: (a) ASCII (full 96, or 64 character subset), (b) EBCDIC, (c) CDC display code, or (d) other character sets if detailed collating sequence is sent.
4. DISTRIBUTION. 7-track magnetic tape (1200 ft. reel) $28.00 (0.8 kg); 9-track magnetic tape (1200 ft. reel) $39.00 (0.8 kg). Subtract $7.00 if you supply a 1200 ft. reel. Longer reels are accepted, but more postage is charged. Overseas orders must add cost of postage and specify type of shipping.
5. DOCUMENTATION. (a) SEG-75-1 "A Preliminary Definition of Janus" $4.00 (180 grams); (b) SEG-76-2 "PASCALJ Implementation Notes" $2.00 (60 grams); (c) SEG-76-1 (*-3?)" Janus Memory Mapping: The J1 Abstraction" $2.25 (60 grams).
6. MAINTENANCE. Every six months (February and September) a new release is planned, but this is subject to manpower constraints. Attempt to fix all reported bugs.
7. STANDARD. (* no information - presumably full Pascal *)
8. MEASUREMENTS. As an interpreter, very slow, but the intent is to do a full bootstrap to a real compiler.
10. DEVELOPMENT METHOD. Compiler originally written in Pascal to generate Janus, and used to translate itself to Janus. Janus processor written in Stage2 macros as an LL(1) system. The set of macros consists of stack operations and indexing in terms of a single accumulator and simple index register. A set of macros for multi-register machines is being implemented. The Stage2 macro-processor is implemented in Fortran.
11. LIBRARY SUPPORT. Not applicable.
PASCAL VARIANTS

Pascal-S

A description of Pascal-S comes from the abstract in the report "Pascal-S: A Subset and its Implementation", by Niklaus Wirth, Institut fuer Informatik, ETH Zuerich, June, 1975. (Available for $6.50 from George Richmond; see address under Pascal-S.)

"Pascal-S is a subset of the programming language Pascal selected for introductory programming courses. This report describes an implementation that is especially designed to provide comprehensive and transparent error diagnostics and economical service for large numbers of small jobs. The system consists of a compiler and an interpreter and is defined as a single, self-contained Pascal program. This machine-independent formulation in a high-level language facilitates its construction and is a prerequisite for easy portability."

Standard Pascal constructs omitted from Pascal-S are: scalar and subrange types, pointers, set and file types, with and goto statements, the passing of procedures and functions as parameters, and several standard procedures. The only file operations are read on input and write on output. The report contains a complete listing of the compiler and interpreter on 34 pages!

Pascal-S is currently distributed on tape with the second release of the CDC-6000 Pascal compiler from Zuerich, Colorado, and Sydney. Pascal-S was implemented in PL/I under Honeywell Multics by the Computer Science Department, University of Southwestern Louisiana, P.O.Box 4-4330, Lafayette, LA 70504 (318/234-7540).

U. Lecarme reported on 7/7/03/04 that Helmut Sandmayr, Neu-Technikum, CH-9470 Buchs, Switzerland (085/6 45 24), has implemented a Pascal-S compiler (not interpreter) for the IBM 1130.

Rich Cichelli reports (7/7/08/31) that an incremental interactive (conversational) Pascal-S compiler was implemented at Lehigh University which is smart enough only to recompile the subprograms in which changes are made.

Concurrent Pascal

A portable pair of Pascal compilers was implemented by Per Brinch Hansen and Al Hartmann at Cal Tech in 1974-1975 for the PDP 11/45. The system consists of a "Sequential Pascal" compiler, a "Concurrent Pascal" compiler (used for writing operating systems and other concurrent programs), and a "kernel" or machine dependent set of run-time routines written in assembler. The project at Cal Tech centered around writing a one-user operating system called SOLO in Concurrent Pascal. Both compilers are written in Sequential Pascal.

In 1975-1976 the system was distributed widely (252 sites) and led to the development of a machine independent version with a different kernel.

As reported in PASCAL News #6, distribution of Concurrent Pascal was terminated in August, 1976, when Per left Cal Tech to join the University of Southern California. On 7/7/07/12, Per phoned to say that distribution may resume and arrangements are being made. Details may be available for Pascal News #11.

Plans are to write a simpler kernel and I/O drivers. This may take 6 months.

Publications about Concurrent Pascal include:

(1) "The programming language Concurrent Pascal", in the June, 1975, IEEE Transactions on Software Engineering 1:2, by Brinch Hansen.

(2) A guest editorial and four articles by Brinch Hansen in the April-June, 1976, issue of Software - Practice and Experience 6, pp 139-205. The articles are entitled:
   - "The Solo Operating System: A Concurrent Program"
   - "The Solo Operating System: Job Interface"
   - "The Solo Operating System: Procedures, Monitors, and Classes"
   - "Disk Scheduling at Compile Time"


Modula

Modula is a small language for dedicated computer systems and process control applications on small machines, developed by Niklaus Wirth and co-workers in 1975-76. It is conceptually cleaner than Concurrent Pascal in many respects. Modula is still experimental and the implementors in Zurich have insisted there are no distribution arrangements. (We are hearing rumors of implementation efforts outside of Zurich though.)

Published material on Modula includes:


(2) The Use of Modula", same as (1), pages 37-65, by Niklaus Wirth.

(3) "Design and Implementation of Modula", same as (1), pages 67-84, by Niklaus Wirth.


The following is the abstract from reference (4), above:

"Programming is divided into three major categories with increasing complexity of reasoning in program validation: sequential programming, multiprogramming, and real-time programming. By adhering to a strict programming discipline and by using a suitable high-level language molded after this discipline, the complexity of reasoning about concurrency and execution time constraints may be drastically reduced. This may be the only practical way to make real-time systems analytically verifiable and ultimately reliable. A possible discipline is outlined and expressed in terms of the language Modula."

Copyright (c) 1977, Association for Computing Machinery, Inc. Reproduction privileges granted by permission of the ACM.
Introduction
I have recently been examining a number of PASCAL programs that are thought by their authors to be highly portable. It rapidly became obvious that it is not realized by the PASCAL community just how many problems are caused by the different character sets used on the computers we have available, nor how this problem is compounded by the set type in PASCAL. This note sets out to make the problems more widely known, and to make recommendations to implementors and programmers.

Character set collating order
There are two very common character sets in the computing industry: EBCDIC (adopted by IBM, Burroughs and ICL 2900 range), and ASCII (adopted by a number of other mainframe suppliers, and most minis), together with a few manufacturers who use their own idiosyncratic character sets (the key example being CDC). In this lot, we can assume nothing about the collating order except that the alphabets collate in ascending order; that the digits collate in ascending order and have successive ORD values; and that the lower-case alphabet collates either lower or higher than the entire upper-case alphabet (if it exists). Practically every other variant of ordering occurs.

This has always been a severe problem to programmers attempting to write portable software, and the advice that can be given only alleviates the problem; it cannot solve it.

Recommendation 1: to PASCAL implementors
All PASCAL compilers should be able to handle objects of type char as internally represented in either the ASCII or the EBCDIC codes, and preferably both. It may be necessary to determine the char representation by a compiler option.

Recommendation 2: to PASCAL programmers
Programmers writing code that depends on the collating sequence of objects of type char should
(i) attempt to collect all such uses into a few routines, and
(ii) adequately comment such uses so that the intent of the code is clear.
This advice applies particularly to programs which process PASCAL text by lexically analysing it.

Available characters
Except that PASCAL originated in CDC machines this would not be a severe problem, since ASCII and EBCDIC have a high degree of commonality in the graphics. Programmers should however be aware that the characters which can be assumed to be available on all computers are limited to the 48 FORTRAN characters. Others are available with varying degrees of probability (for example '>' and ':' are quite highly probable, but '{' and '}' are extremely unlikely). The point of this is mainly felt when designing a language or sub-language or a reply system. An inappropriate choice of character may mean that there is no suitable alternative in another system, and doublet symbols will have to be used (as for example happened with the ( ) and (" ) in PASCAL itself).

The second major deficiency in awareness occurs in respect of the lower-case alphabet. Programmers, through long conditioning, are very proficient at reading solid upper-cased text. The general populace are not, and even programmers read normal text faster and more accurately than the upper-case we normally print. It is thus regrettable that many programs are written so as to totally ignore the existence of lower-case. Programmers should make provision for systems that can read and print lower-case alphabets to use them, even if their system cannot, by simply providing the hooks and commentary.

Recommendation 3: to PASCAL programmers
Be aware of the essential differences between the printable graphics (and in some cases the control characters), and make allowances for these differences. They are important.
The set of char

The PASCAL set construct looks at first sight as to be heaven-sent to enable programmers to write code which is independent of character set collating order. The in operator allows testing a character for membership in a set, rather than having to do relational comparisons.

Alas, this is an illusion. Though conceptually the set construct is ideal, and it is excellent for writing such constructs for sets of more limited size, it falls down badly when it comes to a set of char.

The problem arises because sets are limited in most PASCAL systems to being contained in one or two machine words. Consequently, the size of the set is too small to contain all objects of type char in all the systems I have been able to see. The magnitude of the gap varies, and as PASCAL 6000 comes close to meeting the requirements for a set of char, PASCAL 6000 programmers assume it to be available on all computers.

To illustrate this, let me give the data I have on the set size and on the character set size for various implementations. I regret not knowing any implementation that has a true set of char, but probably one exists. I'd welcome any data on other implementations giving the character set and details as given here.

<table>
<thead>
<tr>
<th>COMPILER</th>
<th>SET SIZE</th>
<th>CHAR SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDC PASCAL 6000</td>
<td>59</td>
<td>63 (CDC special)</td>
</tr>
<tr>
<td>ICL 1900</td>
<td>96</td>
<td>?</td>
</tr>
<tr>
<td>ICL 2900</td>
<td>48</td>
<td>64 (modified ASCII)</td>
</tr>
<tr>
<td>Burroughs B6700</td>
<td>48</td>
<td>256 (EBCDIC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128 (ASCII)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(anysize planned)</td>
</tr>
</tbody>
</table>

The end-effect of this is that set of char is unreliable in CDC machines and virtually not available in other computers. Consequently programs which use this construct are highly unportable. Since the construct (if used) is likely be be used in many places around the program, it then causes considerable difficulty in rewriting the program.

Recommendation 4: to PASCAL implementors

A set of char should give a compile-error unless the whole of the character set can participate as members of the set.

Recommendation 5: to PASCAL implementors

If possible, implementations should permit a maximum set size which will accommodate all characters in the character set.

The main problems centre around set operators, and the creation of set temporaries, if the wordsize is too small. If necessary, large sets may be restricted to the single case of 'set of char'.

Recommendation 6: to PASCAL programmers

That despite its abstract attractiveness, programmers do not write code that contains a set of char anywhere in it.

It might be remarked that there are some programs which are double offenders in the portability stakes; those which gaily use the subrange construct in a set! Thus:

if in ['x' .. 'z'] then ....

Alternatives

Having suggested that the set of char is at present a very poor type to use in a PASCAL program, I ought to indicate some alternatives. While these may lose somewhat in efficiency, it must be borne in mind that portability always has its penalties, and also that efficiency in speed usually only matters in a few critical parts of a program.

The first obvious alternative is to replace each in test (the most usual construct) by a boolean function. The machine-dependencies (if they exist) are then confined to a few places which may be well documented, and are easy to change. An alert programmer might even supply alternatives specialized for a particular computer (like the interchangeable camera lens market).

The second alternative is to examine the uses the construct is put to and see whether or not the requirement is to classify the character into one of a small number of classes (for example: alphabetic, digit, operator, etc). The desired effect may then be achieved by either a function that returns the scalar type value corresponding to the character, or an array might be set up to give the class when indexed by a char. Regrettably, PASCAL does not allow the setting up of read-only arrays, and this will have to be done in a machine-dependent initialization routine.

The first obvious alternative is to replace each in test (the most usual construct) by a boolean function. The machine-dependencies (if they exist) are then confined to a few places which may be well documented, and are easy to change. An alert programmer might even supply alternatives specialized for a particular computer (like the interchangeable camera lens market).

The second alternative is to examine the uses the construct is put to and see whether or not the requirement is to classify the character into one of a small number of classes (for example: alphabetic, digit, operator, etc). The desired effect may then be achieved by either a function that returns the scalar type value corresponding to the character, or an array might be set up to give the class when indexed by a char. Regrettably, PASCAL does not allow the setting up of read-only arrays, and this will have to be done in a machine-dependent initialization routine.
Examples:

```
type
  charclass = (alphabetic, numeric, operator, point, other);
var
  classvector : array[char] of charclass;
function
  classify(ch:char) : charclass;
begin
  case ch of
    'A'..'Z', 'a'..'z', (lattine) 'Y', 'y':
    classify := alphabetic;
    '0', '1', '2', '3', '4', '5', '6', '7', '8', '9':
    classify := numeric;
    '+', '-', '*', '/':
    classify := operator;
    ',':
    classify := point;
    else: (non-standard PASCAL)
      classify := other
  end; (of case )
end; ( of classify )
```

```
begin
  if classify(nextchar) = numeric then ....
  while classvector[nextchar] in [alphabetic, numeric] do ....
end.
```

Set size

The related question of what set size can be reliably assumed to be available is very difficult to answer. I would assume that 32 bits would be safe enough for large/medium computers (the usual word sizes being 32, 36, 48 & 60 bits), but mini-computers pose more of a problem. Diffidently, I suggest that 32 bits be regarded as the minimum set size limit for a compiler to be considered as implementing a compatible PASCAL. Most minis can do this with a double-word. Sets larger than this should be clearly marked in the commentary of a supposedly portable program.

Arthur Sale
Professor of Information Science
University of Tasmania
(Burroughs 86700 implementor)
Undefinition

The definition (and the preceding paragraph) state that after the execution of a for statement (provided the statement is not left by a goto) the value of the control variable is undefined. The primary purpose of this undefinition is to allow implementors freedom to implement the loop efficiently. PASCAL programmers should not therefore presume any particular value in the control variable after it has been used in a for statement. Of particularly nasty characteristics are the compilers which may leave it set at succ(e2), since this may be out-of-range of its type.

In the Burroughs B6700/B7700 computers, it is easy to prevent programmers from doing any computation with this variable until it has been re-defined by setting it to a tag-six word (uninitialized operand). This value can be overwritten by a legal value, but causes a machine interrupt if the variable is used in an operator context. This is done for all for statements in B6700/B7700 PASCAL, and the illegal use of the variable cannot therefore be permitted.

It should also be pointed out that the definition of a for statement allows the control variable to be undefined whether or not the body of the loop is ever entered. B6700 PASCAL treats both cases the same, unlike some other compilers which take advantage of the implementation freedom to leave the control variable unchanged, or at e1, if the loop body is never entered.

Internal change to the control variable

The User Manual, on p24, explicitly forbids the alteration of the control variable by any statement in the body of the loop. Such illegal constructs are hard to detect as they may occur in the body of a procedure or function. On the B6700 computer it is possible to detect this occurrence at run-time with a small time penalty, under some circumstances.

If the loop is capable of being optimized to use the STEP-INDEX facility (which implies that e1 and e2 are in the range 0..65535, and the loop is a TO-loop), then a STEP-INDEX-WORD (SW) is stored in the control variable v. All read-accesses of v return the (integer) value of v, without the final or increment fields, but write-accesses destroy the tag field. Thus when the loop incrementation point is reached, the STBR instruction causes abnormal termination of the loop, and a call is made to the PASCALError routine to kill the program.

The detection facility is not available if the loop is a DOWNTO loop, or if it cannot be simply optimized.

Summary

In this case, and others, the B6700/B7700 PASCAL compiler enforces strict adherence to standard PASCAL. Hardware checks make this possible with negligible time penalty. Programs written in B6700 PASCAL therefore have a higher probability of being portable in this respect than would be the case for many other PASCAL compilers. There is one unfortunate effect, however: non-standard PASCAL programs are less likely to execute in B6700 PASCAL since it is such a searching test.

Implementors of PASCAL are invited to send me answers to the following questions about their compilers. The invitation is also extended to users as implementors are notoriously unreliable corresponents.

1. In what order are e1, e2, and the assignment of v carried out? Does this differ with the form of the loop?
2. What value is left in v if the loop is never entered?
3. What value is left in v if the loop is entered?
4. What happens if the control variable is altered (or a limit-variable)
   (a) from a piece of code compiled in the body, and
   (b) from a procedure called in the body?
5. Are there different (optimized) forms of for-statement? How do they differ?
6. Are there any limits on the number of repetitions or size of the limit-expressions?

If code-templates could be attached (with explanations) this might be useful too. If sufficient information is received, it may be possible to prepare a summary for PUGN.

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

PAGE 68

6700 FOR STATEMENT CODE TEMPLATES

CASE 1 (unoptimized)

\[ a_1 \]
\[ \text{NAMC(TEMP)} \]
\[ \text{STOD} \]

\[ \text{STON} \]

\[ \text{NAMC(V)} \]

\[ \text{STO} \]

\[ \text{TL} : \]
\[ \text{VALC(TEMP)} \]
\[ \text{LSEQ} \]
\[ \text{BRFL(EL)} \]
\[ \ldots \]
\[ \text{body} \]
\[ \ldots \]
\[ \text{VALC(V)} \]
\[ \text{ONE} \]
\[ \text{ADD} \]
\[ \text{NAMC(V)} \]
\[ \text{STON} \]
\[ \text{BRUN(TL)} \]

CASE 2 (unoptimized, but was a potential case for optimization)

\[ a_2 \]
\[ \text{NAMC(TEMP)} \]
\[ \text{STOD} \]

\[ \text{LIT(a2)} \]

\[ \text{NAMC(V)} \]

\[ \text{STO} \]

\[ \text{TL} : \]
\[ \text{VALC(TEMP)} \]
\[ \text{LSEQ} \]
\[ \text{BRFL(EL)} \]
\[ \ldots \]
\[ \text{body} \]
\[ \ldots \]
\[ \text{VALC(V)} \]
\[ \text{ONE} \]
\[ \text{ADD} \]
\[ \text{NAMC(V)} \]
\[ \text{STON} \]
\[ \text{BRUN(TL)} \]

CASE 3 (constant bounds)

\[ \text{LIT(1/25/a1)} \]

\[ \text{NAMC(V)} \]

\[ \text{STO} \]

\[ \text{TL} : \]
\[ \text{VALC(TEMP)} \]
\[ \text{LSEQ} \]
\[ \text{BRFL(EL)} \]
\[ \ldots \]
\[ \text{body} \]
\[ \ldots \]
\[ \text{VALC(V)} \]
\[ \text{ONE} \]
\[ \text{ADD} \]
\[ \text{NAMC(V)} \]
\[ \text{STON} \]
\[ \text{BRUN(TL)} \]

CASE 4 (first bound constant, second capable of optimization)

\[ \text{LIT(1/25/a1)} \]

\[ \text{NAMC(V)} \]

\[ \text{STO} \]

\[ \text{TL} : \]
\[ \text{VALC(TEMP)} \]
\[ \text{LSEQ} \]
\[ \text{BRFL(EL)} \]
\[ \ldots \]
\[ \text{body} \]
\[ \ldots \]
\[ \text{VALC(V)} \]
\[ \text{ONE} \]
\[ \text{ADD} \]
\[ \text{NAMC(V)} \]
\[ \text{STON} \]
\[ \text{BRUN(TL)} \]

\[ \text{ZERO} \]
\[ \text{LIT(6)} \]
\[ \text{STAG} \]
\[ \text{NAMC(V)} \]
\[ \text{STO} \]

\[ \text{TL} : \]
\[ \text{VALC(TEMP)} \]
\[ \text{LSEQ} \]
\[ \text{BRFL(EL)} \]
\[ \ldots \]
\[ \text{body} \]
\[ \ldots \]
\[ \text{VALC(V)} \]
\[ \text{ONE} \]
\[ \text{ADD} \]
\[ \text{NAMC(V)} \]
\[ \text{STON} \]
\[ \text{BRUN(TL)} \]

\[ \text{ZERO} \]
\[ \text{LIT(6)} \]
\[ \text{STAG} \]
\[ \text{NAMC(V)} \]
\[ \text{STO} \]

\[ \text{TL} : \]
\[ \text{VALC(TEMP)} \]
\[ \text{LSEQ} \]
\[ \text{BRFL(EL)} \]
\[ \ldots \]
\[ \text{body} \]
\[ \ldots \]
\[ \text{VALC(V)} \]
\[ \text{ONE} \]
\[ \text{ADD} \]
\[ \text{NAMC(V)} \]
\[ \text{STON} \]
\[ \text{BRUN(TL)} \]

\[ \text{ZERO} \]
\[ \text{LIT(6)} \]
\[ \text{STAG} \]
\[ \text{NAMC(V)} \]
\[ \text{STO} \]

\[ \text{TL} : \]
\[ \text{VALC(TEMP)} \]
\[ \text{LSEQ} \]
\[ \text{BRFL(EL)} \]
\[ \ldots \]
\[ \text{body} \]
\[ \ldots \]
\[ \text{VALC(V)} \]
\[ \text{ONE} \]
\[ \text{ADD} \]
\[ \text{NAMC(V)} \]
\[ \text{STON} \]
\[ \text{BRUN(TL)} \]

\[ \text{ZERO} \]
\[ \text{LIT(6)} \]
\[ \text{STAG} \]
\[ \text{NAMC(V)} \]
\[ \text{STO} \]

\[ \text{TL} : \]
\[ \text{VALC(TEMP)} \]
\[ \text{LSEQ} \]
\[ \text{BRFL(EL)} \]
\[ \ldots \]
\[ \text{body} \]
\[ \ldots \]
\[ \text{VALC(V)} \]
\[ \text{ONE} \]
\[ \text{ADD} \]
\[ \text{NAMC(V)} \]
\[ \text{STON} \]
\[ \text{BRUN(TL)} \]

\[ \text{ZERO} \]
\[ \text{LIT(6)} \]
\[ \text{STAG} \]
\[ \text{NAMC(V)} \]
\[ \text{STO} \]
NOTE TO PUGN

INTERIM REPORT - IMPLEMENTATION OF FOR-STATEMENT - I

The note gives some comparative details on the implementation of for-statements in two PASCAL compilers. As more information becomes available, it will be added to the list. See my earlier comments in a Note to PUGN on the Burroughs B6700 implementation.

BASIC TEMPLATE

for v:=e1 to e2 do s;

PASCAL-6000 (CDC Cyber range)
The implementation produces code which is equivalent to the following:

let temp1 = a register;
    temp2 = a temporary stack location;
    temp1:=e1;
    temp2:=e2;
    while temp1 < temp2 do begin
        v:=temp1;
        s;
        temp1:= v+1;
    end;

The consequences of this code on the precise action of the loop with the three questions I posed are:

(i) the two expressions are computed before an assignment, so that v:=1; for v:=v+1 to v+10 do s; will count from 2 to 11.
(ii) The exit value of v if the loop is never entered is its value before the loop is reached.
(iii) The exit value of v is e2 if the loop is ever traversed.

In addition, alterations of v from within the body of the loop do in fact alter the progress of counting, if they can be achieved.

PASCAL for Burroughs B6700/B7700 (Tasmania)
More details are given in the Note mentioned before. The code is generally equivalent to:

let temp1 = a temporary stack location;
    temp2 = a temporary stack location;
    temp1:=e1;
    temp2:=e2;
    v:=temp1;
    while v < temp2 do begin
        s;
        v:=v+1;
    end;
    v:=invalidtagsixvalue;

The answers are again:

(i) as for PASCAL-6000.
(ii) + (iii) In all cases the exit value of v is a special word which cannot be utilized as a value, but can be overwritten with a proper value.

Arthur Dale

Implementation Note

1977 February 17

P6700/B7700 PASCAL : ELSE IN CASE

Introduction
Many PASCAL implementations are inserting an ELSE clause in the CASE statement of PASCAL. This note puts the cases for and against, and proposes a pseudo-standard for any such implementations so that maximum compatibility between PASCAL compilers can be achieved.

Against
The case against having an ELSE clause in a case statement is that it encourages a programmer to use the clause through laziness simply to save writing a long list of alternatives. Thus when an unexpected value of the case expression occurs, it is processed erroneously by the ELSE clause, rather than being one of the 'undefined' areas of PASCAL. The arguments here rest on Implementors choosing to detect values of the case expression which do not match any label, and choosing to make such occurrences definite run-time errors. Such an interpretation is not mandatory.
For

The arguments for an ELSE clause are regularity, and robustness. The regular argument comes from (i) examination of languages of similar age and utility, in most of which the feature appears, (ii) the analogy with if-then-else which may be viewed as a special version of case, and (iii) actual thought habits of good programmers.

The robustness argument derives from the need to be able to write programs which are robust against all input, and all circumstances, and from the difficulty of handling all case statements without error. Long lists of labels are error-prone, and sometimes inappropriate. If the intention is that all values other than a specified few are to be similarly treated, then it ought to be possible to specify this.

The B6700 implementation

The implementation of PASCAL for the Burroughs B6700/B7700 computers developed at the University of Tasmania contains such an ELSE facility. The semantic features of this implementation are suggested as a pseudo-standard for PASCAL implementors who also agree that this is a necessary feature.

A case without else

If no else appears in a case statement, the B6700 implementation will raise a run-time error event, and terminate the program, if the case expression evaluates so as to match no case label.

Recommendation 1
That all implementations of PASCAL regard the above as the preferred semantics of this situation arising in a case statement.

A case with else

If an else clause appears in a case statement, then the B6700 implementation transfers control to the else clause for all values of the case expression which do not match an explicit case label. In all other respects an else clause behaves as a labelled clause.

Recommendation 2
That the above be regarded as the minimum semantic requirements of an else-clause in a case-statement. If an implementation can cause the same effect as in Recommendation 1 for values of the case-expression which are outside its declared range (as in the type), they are encouraged to do so. This is relevant only to implementations that include an else-clause.

Syntax of else-clause

In the B6700 implementation, an ELSE can appear wherever a case-label can, except that there can be at most one in any case statement. Thus an ELSE may appear in a case-label list, though it is difficult to see why this would be done. This syntax is very easy to accommodate, and requires minimal changes to the CASESTATEMENT routine in PASCAL-P4. There are no other syntactic changes.

Recommendation 3
If an implementation adopts an ELSE-clause, then the above syntax should be regarded as standard. Modified syntax diagrams are attached.

Stylistics

The preferred style for a case statement containing an else clause has the else clause last, following all labelled clauses.

Example of case with else

case of
  'a', 'b', 'c', '/' :
    thing := arithmeticoperator;
  ';', ',', '[', ']' :
    thing := variablelocator;
  '(', ')', 'i', 'i' :
    thing := separator;
else:
  thing := otherthing
end;

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

MODIFIED SYNTAX CHART FOR CASE-STATEMENT IN WIRTH-FORM
The impression that variable-parameters in Pascal must be passed by reference is widespread (e.g. it appears in the books by Conway, Gries and Zimmermann and by Webster). However, I believe it to be a misconception stemming from the fact that all existing implementations have used reference passing. Many other controversies in the Pascal Newsletter arise from this failure to distinguish between language and implementation. My understanding of the matter is that (as in Fortran and for the same reason) both reference and value-result are valid mechanisms for variable-parameters.

If we look at Section 9.1.2 of the Report we find only that the formal "represents" the actual during the execution of the procedure. Name binding is disallowed (thank heavens!) by the rule that the index of a subscripted variable-parameter is evaluated just once, but reference is not specified.

In the Axiomatic Definition, at axiom 11.2, it is stated that the variable-parameters and non-local variables accessible by a procedure-call must be distinct (no "aliasing"). Given this condition, it is not possible to determine the parameter-passing mechanism by running a legal program. I conclude that any method which satisfies axiom 11.2 is allowable.

This issue is not just of theological interest. The implementor has been given an important degree of freedom: he can copy the technique used by the Fortran system on his machine and thereby gain access to the enormous investment in Fortran library routines.

---

**Interactive with a PASCAL program** - D. A. Joslin, University of Sussex, Computer Centre, Brighton, U.K.

The requirement of the Revised Report that COMPUTE be defined right from the very start of a program (e.g. generally: COMPUTE is defined immediately after READ(j), and READ(f,x) = x = f(x); SET(f) ) results in the first card being physically read into a buffer when the program is entered, the second card being physically read on the first READIN, and so on. An interactive program, however, normally outputs some message to the terminal before expecting the user to type his first input: it proceeds in a question/answer/response cycle. This can be achieved in PASCAL provided that:

(i) the operating system is instructed to satisfy the program's first card-read request by any answer record (which the program will not actually process), and second and subsequent requests by terminal input;

(ii) the program precedes each READ, ie each request for an answer from the terminal user, by a READIN. This can conveniently be done by means of procedures - eg

```
PROCEDURE GENERAL(VAR X:REAL);
BEGIN READIN; READ(X) END;

...............;

UNFORMAT( 'LETTER A REAL NUMBER' ); (* question *)
GENERAL(X); (* get answer *)
WRITEOUT( X =", X); (* response to answer *)
```

The attached sheet shows a sample program COEXIST written according to rule (ii) above:

a George 3 macro INTERACT which performs the action of rule (i) above - the command INTERACT is given by the terminal user after he has loaded a program he is to interact with; a sample teletype session showing interaction with COEXIST via INTERACT.
PASCAL COMP, OBJECTCC21INTPROG
TYPE STRING = PACKED ARRAY [1..16] OF CHAR;
VAR T: STRING; J,K: INTEGER; E: CHAR;

PROCEDURE GETSTRING(VAR T: STRING);
BEGIN
  READLN(E);
END;

PROCEDURE GETINTEGER(VAR I: INTEGER);
BEGIN
  WRITE('PLEASE TYPE IN YOUR NAME: ');
  READLN(T);
END;

BEGIN
  WRITE('HI THERE! - PLEASE TYPE IN YOUR NAME
        AND A WHOLE NUMBER.
        YOUR NUMBERS ARE 123 AND 77.
        THEIR SUM IS 190 AND THEIR DIFFERENCE IS 46
        SHALL WE TRY AGAIN?
        - YES
        - 1
        AND ANOTHER
        - 5
        YOUR NUMBERS ARE 77 AND 1
        THEIR SUM IS 78 AND THEIR DIFFERENCE IS -74
        SHALL WE TRY AGAIN?
        - YES
        - 5
        AND ANOTHER
        - 95
        YOUR NUMBERS ARE 85 AND 96
        THEIR SUM IS 181 AND THEIR DIFFERENCE IS 104
        SHALL WE TRY AGAIN?
        - NO
  
  GREETING: DAVID

  GLAD TO MEET YOU, DAVID
  PLEASE TYPE IN A WHOLE NUMBER
  - 153
  AND ANOTHER
  - 57
  YOUR NUMBERS ARE 153 AND 57
  THEIR SUM IS 160 AND THEIR DIFFERENCE IS 96
  SHALL WE TRY AGAIN?
  - YES
  TYPE IN A WHOLE NUMBER
  - 1
  AND ANOTHER
  - 39
  YOUR NUMBERS ARE 1 AND 39
  THEIR SUM IS 40 AND THEIR DIFFERENCE IS -38
  SHALL WE TRY AGAIN?
  - YES
  TYPE IN A WHOLE NUMBER
  - 5
  AND ANOTHER
  - 77
  YOUR NUMBERS ARE 5 AND 77
  THEIR SUM IS 82 AND THEIR DIFFERENCE IS -72
  SHALL WE TRY AGAIN?
  - NO

  EXIT
END.

***

#27
PASCAL Compiler #PASO/3A (SUSSEX VERSIN #001) ON 06/05/77 AT 11:49:44
OPTION(S) SELECTED: NOFL

19-19-51- LISTFILE INTERACT, NUMBERS
0 *MACRO INTERACT - B.A. JELIN, MAY77
1 (L *LP)
2 OK
3 IN 1, ??
4
5 ??
6 AS *CH01
7 OK
8 RP *FP+LS+OL+PM
9 OK
10 IF FAIL(FILE *CR0), JL *CD0
11 RP *PM, DM
12 OK

19-20-52- LOAD CCRTINTPROG.CREXK
19-21-05- INTERACT
Hi THERE! - PLEASE TYPE IN YOUR NAME
- DAVID
Glad to meet you, DAVID
Type in a whole number
- 153
And another
- 57
Your numbers are 153 and 57
Their sum is 160 and their difference is 96
Shall we try again?
- Yes
Type in a whole number
- 1
And another
- 39
Your numbers are 1 and 39
Their sum is 40 and their difference is -38
Shall we try again?
- Yes
Type in a whole number
- 5
And another
- 77
Your numbers are 5 and 77
Their sum is 82 and their difference is -72
Shall we try again?
- No

Ending dialog...
MACHINE DEPENDENT IMPLEMENTATIONS

---Pascal Implementations Summary---

(* This section summarizes all the information that we have on all Pascal implementations, in the checklist format. *)

Andahl 470

(* See implementation notes for IBM 360/370. *)

Burroughs B1700

In a letter dated November 3, 1976, Tony Gerber (Basser Dept. of Computer Science, School of Physics, University of Sydney, Sydney, N. S. W. 2006, Australia; Tel. 629 1122) reported several persons who have worked on B1700 implementations. They are:

Elliott Organick's group at the University of Utah, using Brinch Hansen's Sequential Pascal.

P. Schultess and K. Hauserman at the University of Zuerich, who each worked on (separate) projects.

P. Albrich, University of Karlsruhe, Germany, was working with Concurrent Pascal.

M. Ellison at the University of Newcastle-upon-Tyne was using Pascal-P "version 1.0".

Burroughs B3700, B4700

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. R. M. Lansford; 3620 Greenhill Rd.; Pasadena, CA 91107; 213/351-0206. F. L. McCullough; Tektronix 60/666; P.O. Box 500; Beaverton, OR 97077 (503/638-3411 x2397). W. C. Price; 28282 SW Mountain Road; West Linn, OR 97068 (503/644-0161).

2. MACHINE. Burroughs B3700, B4700 (with Accumulator operator.)

3. SYSTEM CONFIGURATION. MCVP 5.7 and Time Sharing System

4. DISTRIBUTION. No plans at present--the need has not arisen.

5. DOCUMENTATION. Foreward to program listing; in form of supplement to Pascal User Manual and Report. (* This is apparently not machine retrievable. *)

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

7. STANDARD. Extensions:

real arithmetic
formal procedures and functions
files (except text files INPUT and OUTPUT)

segmentation
symbolic procedure call tracing
stack checking and statistics
packing is optional

--- 8. MEASUREMENTS.

Pass 1: 4000 lines of Pascal, compiled @ 1000 lines/min.
Pass 2: 2500 lines of BFL, taking 45 sec. to generate code for Pass 1 of the compiler.
A minimum of 110K bytes is needed for a logical (reasonable) segmentation of the compiler. (* Size and execution speed of code produced not reported. How this compares to FORTRAN and other languages not reported. *)

9. RELIABILITY. Good, but not excellent. (* Number of sites using compiler not reported. Date first released not reported. *)

10. DEVELOPMENT METHOD. Compiler was bootstrapped from an early PI compiler obtained from Caltech. The compiler consists of two passes. The first is written in Pascal and emits augmented P-code. The second pass (written in BFL, a PL/360-like assembler) generates 4700 code from the P-code. The first version of the compiler was written by Mike Mahon in 2 person-months. An additional 8 person-months have been expended in teaching the compiler about such things as optimal variable size and alignment, segmentation, etc.

11. LIBRARY SUPPORT. (* No information provided. *)

June 8, 1977

Mr. Andy Mickel
PASCAL User's Group
University Computer Center
227 Experimental Engineering Building
University of Minnesota
Minneapolis, Minnesota 55455

Dear Andy:

Thank you for the incredible amount of effort you have put into making PUG work. Please, however, don't use anymore of that ugly chartreuse paper.

As to the Burroughs B3700/B4700 PASCAL implementation reported by Dr. Lansford in PUGN#8: Due to the efforts of Burroughs' management, the (spare-time) project has been cancelled. We understand that inquiries through Burroughs Medium Systems Plant have been answered with "Ask your local Burroughs representative." The reports we promised on certain interesting aspects of our implementation (segmentation, optimization, augmentation of P-code, etc.) have been delayed (perhaps indefinitely), as we are no longer associated with Burroughs Corporation.

Herewith, however, is a short comment arising from our attempt at understanding the full implications of PASCAL's file structure.

Truly,

William C. Price
Instrument Research Group
Tektronix Laboratories

WCP:pt
Attachment
cc: Dr. R.M. Lansford
F.L. McCullough
Burroughs B5700

Bruce A. Pumplin, Department of Computer Science, University of Wisconsin - Eau Claire.
Eau Claire, WI 54701, has promised us a report on the progress of his Pascal-P based
implementation for the B5700. Last we knew (7/6/80), the compiler-interpreter was
working.

Burroughs B6700/7700 (Tasmania)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. A.J.J. Sale; Dept. of Information Science;
University of Tasmania; Box 252C G.P.O.; Hobart, Tasmania 7001 Australia; STO 002 23-0561
x453.
2. MACHINE. Burroughs Model III B6700, B7700
3. SYSTEM CONFIGURATION. Burroughs MCP version 11.8 (with few (minor) local mods).
Minimal system to operate not known, but unlikely to be any B6700 that small—storage
demands are low, and little else is critical.
4. DISTRIBUTION. Both 7 and 9 track magnetic tapes available. (* Cost not reported, *)
5. DOCUMENTATION. Supplement to Pascal User Manual and Report available; a dictionary-
style Reference Manual is in preparation but is not yet complete (7/7/4/40). (* Not known
if this documentation is machine retrievable, *)
6. MAINTENANCE. To be maintained for teaching use within the University as well as larger
aims. Reported bugs will be fixed as soon as possible, with patch notices to users. Duration of support not yet determined; several other developments are also pending.
7. STANDARD. Restrictions: Program heading: reserved word program is synonymous with
procedure; no parameters (files) are permitted after the program heading. Reason: CDC

8. MEASUREMENTS. Compiles about 20% slower than Fortran or Algol, but in about 2/3 of
their space (for test programs about 4-5 K words on average instead of 8-10K). Elapsed compilation times similar, though Pascal slower. Speed should be improved by eventual tuning. Executes at same speed as Fortran and Algol (code is very similar and optimal) and takes generally longer elapsed residence time primarily due to MCP intervention to create new segments for record structures (not present in Fortran/Algol). Elapsed residence times about 20% greater than equivalent Algol.
9. RELIABILITY. Excellent. Only one system crash during testing attributed to Pascal.
Compiler now in use at 3 sites. Compiler has been in use since 7/9/70. First released to
outside sites in 7/7/4.
10. DEVELOPMENT METHOD. Compiler which generates B6700 code-files which are directly
executed by the B6700 with MCP. Written entirely in B6700 Algol. Hand-coded using Pascal-P
as a guide/model. All other paths offered much more difficulty due to special nature of
machine/system. Person-month details not kept, and project proceeds in fits and starts as
teaching intervenes. Project has thus far been limited to two people: Prof. A.H.J. Sale
and R.A. Freak (support programmer).

11. LIBRARY SUPPORT. There is as yet no BINDINU in the code-file so that it is not
possible to link Pascal to modules compiled by other language processors, but the system
contains an extended set of predefined mathematical functions.

Burroughs B6700 (San Diego)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Distributor: Henry Fischer; UCSD Computer Center;
University of California - San Diego; La Jolla, CA 92039; 714/452-4030. Implementors: Mark
Overgaard; Jim Hadden: same site.
2. MACHINE. Burroughs B6700
3. SYSTEM CONFIGURATION. (* No information provided, *)
4. DISTRIBUTION. Scheduled to start in mid-summer, 1977. (* Information on cost, magnetic
tape formats, etc. was not provided. *)
5. DOCUMENTATION. (* No information provided, *)
6. MAINTENANCE. Unknown at this time.
7. STANDARD. (* No information provided. *)
8. MEASUREMENTS. Current compile speed is 5000 line/min; but expected improvements could
make that 10,000 lines/min--as fast as the Burroughs Fast Algol Compiler. (* Size and
execution speed of code produced not reported. *)
9. RELIABILITY. Unknown at this time. (* Number of sites using this compiler not
provided. Date of first release not reported. *)
10. DEVELOPMENT METHOD. Real compiler, written in Pascal which produces native code for
the B6700. (* Person-hours to create compiler not reported. *)
11. LIBRARY SUPPORT. (* No information provided. *)

Burroughs B6700 (New Zealand)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Chris Bishop; Computing Centre; University of
Otago; P. O. Box 56; Dunedin; NEW ZEALAND; (Tel. Dunedin 40109 x890).
2. MACHINE. Burroughs B6700
3. SYSTEM CONFIGURATION. (* No information provided. *)
4. DISTRIBUTION. Tapes can be written in any of the following formats:
   a) 1600 bpi, 9K, track, B6700 library tape
   b) 800 bpi, NRZ, 9 track, B6700 library tape
   c) 1600 bpi, PE, 9 track, USASI Multi-file tape
   d) 800 bpi, NRZ, 9 track, USASI Multi-file tape
(* Costs for tapes not reported. *)
5. DOCUMENTATION. Brief notes on usage available. (* Not known if this is machine
retrievable. *)
6. MAINTENANCE. (* No information provided. *)

7. STANDARD. (* No information provided. *)

8. MEASUREMENTS.
   compilation space-- (* No information provided. *)
   compilation speed--Compiles the Karlsruhe 86700 compiler in 90 sec. of processor time.
   execution speed-- (* No information provided. *)
   execution space-- (* No information provided. *)
   (* How this compares to FORTRAN and other languages not reported. *)

9. RELIABILITY. Unknown at this time. Compiler in use at 3 sites. (* Length of time
   compiler has been in use not reported. *)

10. DEVELOPMENT METHOD. Karlsruhe 86700 compiler-interpreter translated from Pascal source
to Burroughs Extended Algol. 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 Extended Algol. It is
planned to convert this Algol version into a true compiler for the
Burroughs 86700; work will start in earnest about July of 1977. (* Person-hours to create compiler not reported. *)

11. LIBRARY SUPPORT. (* No information provided. *)

Burroughs 86700 (Helsinki)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Antti Salava; formerly at Dept. of Computer
Science; University of Helsinki; Toolonkatu 11, SF-00100; Helsinki 10, Finland; Present
address: Munkkiniemen Pulistatie 17-A13; SF-00330 Helsinki 33, Finland; phone: 90-496288.

2. MACHINE. Burroughs 6700

3. SYSTEM CONFIGURATION. (*Unknown *)

4. DISTRIBUTION. None; project not yet complete.

5. DOCUMENTATION. We are currently (77/1/17) preparing a report on our Pascal
implementation. (* Not known if this will be machine retrievable. *)

6. MAINTENANCE. None, project not yet complete.

7. STANDARD. (* No information provided. *)

8. MEASUREMENTS. Unknown; project not yet complete.

9. RELIABILITY. Unknown; project not yet complete.

10. DEVELOPMENT METHOD. The compiler is written in Burroughs Extended Algol and generates
Burroughs 86700 machine code. (* Person-hours to create compiler not reported. *)

11. LIBRARY SUPPORT. (* No information available. *)

CDC Cyber 18 and 2550

---

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Jim Fontana, Control Data Corporation,
3519 W. Warner Ave., Santa Ana, CA 92704 (714/754-4102).

2. MACHINE. Control Data Cyber 18 and 2550.

---

3. SYSTEM CONFIGURATION. (* The Cyber 18 is a self contained interactive system. *)
Dennis Nicolai (CDC, Minneapolis) reports that the Cyber 18 and the 2550 have similar
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 and 2550.

4. DISTRIBUTION. Control Data Corporation.

5. DOCUMENTATION. CDC Manual 88988500 A. (* Apparently no machine retrievable
documentation available. *)

6. MAINTENANCE. CDC supported Communications Front End software.

7. STANDARD. Unrevised Pascal language definition with extensions. I/O is hardware
defined.

8. MEASUREMENTS. (* No information available. *)

9. RELIABILITY. Excellent. (* Number of sites using system not reported. Date of first
release not reported. *)

10. DEVELOPMENT METHOD. The compiler is derived from the compiler for the CDC 2550
front end processor, which in turn was derived from the old Zurich Pascal-6000 (1972) compiler.

11. LIBRARY SUPPORT. (* No information available. *)

---

CDC 3200

---

A local rumor is that John Urbanski, West Bank Computer Center, 90 Blegen Hall,
University of Minnesota, 269 19th Ave. South, Minneapolis, MN 55455 USA (612/373-3608),
is working on an implementation of a subset of Pascal for the CDC 3200.

CDC 3300

---

We have not heard any news from either of the following two implementors for over two
years, in spite of several attempts by us and others to reach them.

P. J. Voda, Computing Research Centre, Dubravska 3, 885 31 Bratislava, Czechoslovakia, has
a version of Pascal operational on the 3300. This version includes concurrent constructs
(not the same as Brinch Hansen's), and several large software projects were implemented
using it.

Lou Bervino, Computer Center, California State University, Northridge, CA 91324, is
known to have received Pascal-F2.

CDC 3600

---

This is another case of the "two-year silence" (see CDC 3300). You are welcome to try
contacting Marcel Dupras, Institut de Programmation, Tour 55-65, 11-Quai Saint Bernard,
F-75 Paris, France, who was listed by George Richmond as having completed an
implementation on the 3600.
CDC 6000, Cyber 70, Cyber 170 (Zurich)

**1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER.**

Distributors:
- (Europe, Asia, or Africa)
  Ura Ammann
  (same address as implementor *)
  (North or South America)
  George H. Richmond
  Computing Center: 3645 Marine St.
  University of Colorado
  Boulder, CO 80309 USA
  303/ 492-8131
- (Australia, New Zealand, or Oceania)
  Carroll Morgan
  Basser Dept. of Computer Science
  University of Sydney
  Sydney, N.S.W. 2006
  Australia
  629 1122

**2. MACHINE.**

Control Data 6000 series, Cyber 70 series, and Cyber 170 series.

**3. SYSTEM CONFIGURATION.**

Minimum central memory-49K words. Operates under Scope 3.4 and Kronos 2.1.

**4. DISTRIBUTION.**

Tape format is Scope 3.4 internal binary, 7 track, unlabelled, 800 bpi. Specify: person responsible for maintaining the system, your hardware, operating system, and character set (ASCII or Scientific, 63 or 64). From Switzerland cost is SFr. 100 (includes cost of tape; do not pay in advance, you will be billed); from Colorado cost is $60 for new recipients (includes tape and documents), and $35 for old recipients (includes tape but not documents); from Australia cost is $A30 (tape and documents). New installation notes will be machine retrievable in Release 3.

**5. DOCUMENTATION.**

Machine retrievable supplement to Pascal User Manual and Report and documentation of library support package will be available with Release 3.

**6. MAINTENANCE.**

Will accept bug reports at Minnesota for foreseeable future. Expect to issue Release 3 in 1978.

**7. STANDARD.**

Nearly full standard. Restrictions include: standard procedures and functions cannot be passed as actual parameters; file of file is not allowed. Extensions include: additional predefined procedures and functions; segmented files.

**8. MEASUREMENTS.**

Compilation speed: 10500 characters per second on a Cyber 74; 54 seconds to compile the compiler. Compilation size: 46K (octal) words for small programs; 57K for self-compilation. Execution speed: see 7600 statistics, below. Execution size: binaries can be as small as 2.4K, compared with Fortran minimum of over 10K.

**9. RELIABILITY.**

Excellent. The compiler is in use at 139 known sites. First version of this compiler was operational in late 1970. The present version was first released in May 1974.

**10. DEVELOPMENT METHOD.**

Bootstraped from the original Pascal-6000 compiler, but developed in a 6 phase stepwise refinement method. Approximately 1.5 person years.

**11. LIBRARY SUPPORT.**

Allows calls to external Pascal and assembler subprograms and Fortran (FTN) subprograms. The user library supplied with the system contains many routines in addition to the standard.

**CDC 7600, Cyber 76 (Manchester)**

(* See announcement elsewhere in this issue. *)

**1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER.**

This compiler is essentially the Pascal 6000 compiler modified to fit the 7600 and Cyber 76 machines. The run time system is based on that of Hans Jornstad at CERN, Geneva, Switzerland (see Pascal News #4). Improvements by H. D. Ellison; A.P. Hayes; UMRCC; Oxford Road; Manchester M13 9PL; England, U.K.; (061-273 8252).

**2. MACHINE.**

Control Data 7600 & Cyber 76.

**3. SYSTEM CONFIGURATION.**

SCOPE 2.1.3, 32K SCM.

**4. DISTRIBUTION.**

Contact R. J. Collins at address above. A distribution agreement must be signed and the cost is 50 pounds sterling.

**5. DOCUMENTATION.**

Same as Pascal-6000.

**6. MAINTENANCE.**

The situation is unclear at present. UMRCC will assist with bugs — in the 7600 dependent code (runtime system) only. Currently UMRCC and Minnesota will work together on a common release for Release 3.

**7. STANDARD.**

Same as Pascal 6000.

**8. MEASUREMENTS.**

Compilation speed is about 57,000 characters/sec. Compiler compiles itself in less than 10 sec. Pascal execution speed has been measured by using the obvious encoding in Pascal of Wichmann’s Synthesis Benchmark (see Computer Journal Vol. 19, #1).

The units are in kilo Whetstones.

<table>
<thead>
<tr>
<th>Compiler and</th>
<th>no runtime</th>
<th>array bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimisation level</td>
<td>checking</td>
<td>checking</td>
</tr>
<tr>
<td>ALGOL 4 (OPT=5)</td>
<td>1996</td>
<td>1230</td>
</tr>
<tr>
<td>Pascal (OPT=5)</td>
<td>6850</td>
<td>6240**</td>
</tr>
<tr>
<td>FTN (OPT=2)</td>
<td>965</td>
<td>3174**</td>
</tr>
</tbody>
</table>

* Using T+ option—all run time checks included.

** Forces OPT=0.

Compiler will recompile itself on a ‘half-size’ (32K SCM) machine. (* No information provided on size of compiler or object code produced. *)

**9. RELIABILITY.**

3 sites; as reliable as Pascal 6000 (Zurich). (* Date of first release not reported. *)

**10. DEVELOPMENT METHOD.**

Cross compiled from Cyber 72 compiler. Based on Zurich 6000 compiler with necessary additions for this machine. (* Person-hours to develop compiler not reported. *)

**11. LIBRARY SUPPORT.**

Same as Pascal 6000.

CDC Omega 480-I, 480-II

(* See implementation notes for IBM 360/370. *)
Dear Andy:

This is to inform you that a PASCAL implementation has been completed for the CDC STAR-100. The details are:

1. **Implementors:**
   - Douglas B. Dunlop
     - Dept. of Mathematics
     - College of William & Mary
     - Williamsburg, VA 23185
   - John C. Knight
     - Analysis and Computation Division
     - NASA Langley Research Center
     - Hampton, VA 23665

2. **Language:**
   - The PASCAL P4 subset of PASCAL.

3. **Machine:**
   - Control Data Corporation STAR-100.

4. **Operating System:**
   - STAR O/S.

5. **Documentation:**
   - No formal mechanism. Write if you are interested.

6. **Reliability:**
   - Compiler correctly compiles itself.

7. **Distribution:**
   - The compiler was developed from PASCAL P4. Two forms exist and both compilers generate STAR machine code. They are a 6000 based cross compiler which produces object modules for input to the STAR loader, and a STAR resident compile and execute system.

Our experience with PASCAL P4 has been very satisfactory and we congratulate the developers. In less than six man weeks of effort, the PASCAL P4 compiler was modified to generate STAR-100 machine code, and the compiler which was produced successfully compiled itself on the STAR-100.

Sincerely,

John C. Knight
Aerospace Technologist
Programming Techniques Branch

---

1. **IMPLEMENTOR/DISTRIBUTOR/MAINTAINER.** Olivier Lecarme; Universite de Nice; Laboratoire d'Informatique; Parc Valrose, 06034 Nice Cedex; France (51 91 00).
2. **MACHINE.** CII IRIS 50.
3. **SYSTEM CONFIGURATION.** Siris 3 operating system. (*Minimum hardware requirements not known.*)
4. **DISTRIBUTION.** (*Unknown, project not yet complete.*) Expected to be available by end of 1977.
5. **DOCUMENTATION.** (*No information provided.*)
6. **MAINTENANCE.** (*Unknown, project still underway.*)
7. **STANDARD.** Will implement exactly Standard Pascal.
8. **MEASUREMENTS.** (*Unknown, project not yet complete.*)
9. **RELIABILITY.** (*Unknown, project not yet complete.*)

10. **DEVELOPMENT METHOD.** Various approaches tried. Tool compiler developed using Pascal-P, Pascal-E subset, intermediate machine oriented languages, and the Nagel trunk compiler used to write a true compiler. (*Person hours to implement system not reported.*)
11. **LIBRARY SUPPORT.** Will produce modules for the linkage editor. (*No information provided on external and other language subroutines, separate compilation, automatic source inclusion, or symbolic post-mortem dumps.*)
8. MEASUREMENTS.

compilation speed—1800 Pascal lines/min.;
2400 characters/sec. versus
1200 characters/sec. for FORTRAN.
compilation space—run the Pascal system:
30 K words with overlay;
45 K words without overlay.
execution speed—dependent on program profile; compared to FORTRAN:
matrix multiplication 1 1.6
recursive program 1 0.3
character count on file 1 0.2
execution speed—(* No information provided. *)

9. RELIABILITY. Good to Excellent. This is release 3 of this compiler. The compiler has
been tested since 1974 in 30 installations.

10. DEVELOPMENT METHOD. Full compiler generating object code for the linkage editor.
The compiler consists of
a MONITOR: programmed in CII's local assembly language (2K 32-bit words).
It links the Pascal program to the operating system and controls the
execution of the Pascal program. All operating system dependencies
are located in this monitor. To get the compiler available on some
other operating system, the rewriting of this monitor is necessary.

a COMPILER: written in Pascal itself, it consists of 4800 lines. It is a
one pass compiler with top-down syntax analysis, separate compilation
of Pascal programs, symbolic post mortem dumps, and several specific
options. The compiler is fully bootstrapped so that any user may
adapt it easily to a specific need (change the table sizes, specific
features, etc.).

a LIBRARY used by the linkage editor.
(* Person-hours to create compiler not reported. *)

11. LIBRARY SUPPORT. Separate compilation of Pascal programs implemented. (* No
information on subprogram libraries. *)

Computer Automation LSI-2

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Computer Automation; Naked Mini Division; 18651
Von Karman; Irvine, CA 92713; 714/833-8830; TWX:910 595 1767.

2. MACHINE. Computer Automation LSI-2 (16-bit minicomputer).

3. SYSTEM CONFIGURATION. Computer Automation OS. Minimum hardware: moving head or floppy
disk and 12K Memory.

4. DISTRIBUTION. Distributed on floppy disk for $900.

5. DOCUMENTATION. User's Guide explaining the use of Pascal under CA-OS. (* Apparently no
machine retrievable documentation. *)

6. MAINTENANCE. Fully supported including acceptance and response to user trouble
reports. In the near future, standard Pascal I/O will be implemented.

7. STANDARD. Implements Sequential Pascal which varies from standard Pascal. Missing:
reserved words file, goto, label, packed; mixed type arithmetic; standard functions: ODD,
EVEN, EOF, SQRT, ROUND, SIN, COS, ARCTAN, LN, EXP, SQR. Restricted to 2 levels of static
nesting. Has extended I/O and file access methods.

8. MEASUREMENTS. (* No information provided. *)

9. RELIABILITY. Very good. (* Number of sites using system not reported. Date first
released not reported. *)

10. DEVELOPMENT METHOD. Seven pass compiler. (* Method of developing compiler not
reported. Number of person-hours to implement compiler not reported. *)

11. LIBRARY SUPPORT. Automatic formatting option implemented. (* No information
provided on separate compilation or subprogram libraries. *)

CRAY-1 (Los Alamos)

PACIFICA NEWS #9 & 10

SEPTEMBER, 1977

P. O. BOX 1663
LOS ALAMOS, NEW MEXICO 87544

IN REPLY
REFER TO: C-11
MAIL STOP: CUB

July 7, 1977

Dear Andy:

Despite Bob Johnson's rather discouraging letter, (PACIFICA Newsletter #6), PASCAL still lives on
the CRAY-1. We now have a new version based on Sasan Hazen's P-code Post Pro­
gressor concept (W.H. P7). Current plans are to extend P-code and the P-code translator to provide
code generation for the Alpha compiler.

I enclose an 11-point Newsletter-style description of our
implementation, the User's Guide Addendum, and my check
for $4 for next year's P.U.G. membership.

Sincerely,

John Montague

CRAY PASCAL (Version 2)

1. Implementors:

The compiler was bootstrapped by John Montague and
Michael Powell. Many of the code templates were taken
from Bob Johnson's cross compiler. Nearly all changes
and improvements since the bootstrap were completed are
due to Forest Baskett and Linda Zucconi. We can all be
reached at the following address:

Los Alamos Scientific Laboratory
Group C-11, Mail Stop 290
P. O. Box 1663
Los Alamos, New Mexico 87545
(505) 667-7877
2. Machine:
Cray Research, Inc. CRAY-1

3. Operating System:
Benchmark Operating System (BOS), a LASL modified version of the CRAY-CRAY-OS Version 1.

4. Distribution:
Distribution is arranged on an ad hoc basis. All (both) current CRAY-1 installations have a copy.

5. Documentation:
Short write-up on the differences between CRAY PASCAL and Standard PASCAL, plus instructions for use.

6. Maintenance:
We will maintain CRAY PASCAL at LASL as long as we find it useful. The compiler is still undergoing development, and new versions will probably be complete replacements rather than updates. A project is underway to use the P-code translator as a code generator for Model, a LASL-developed language which will be used for much of our new CRAY-1 operating system.

7. Standards:
CRAY PASCAL implements the subset of PASCAL defined by the PASCAL-P compiler with a few extensions toward Standard PASCAL and several of the PASCAL-6000 predefined functions and procedures.

8. Compiler Implementation:
CRAY PASCAL is written in PASCAL and consists of two separate programs, the PASCAL-P compiler (version 2, extended by Sassan Hazeghi of Stanford University to the equivalent of P4, and further modified at LASL) and the P-CODE translator which converts P-CODE into CRAY Assembly Language (CAL). Despite some character set problems, both programs currently run on the CRAY-1 and on a CDC 6600 under NOS. Some statistics on our implementation are:

<table>
<thead>
<tr>
<th>Lines of source code</th>
<th>P-compiler Translator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4800</td>
</tr>
<tr>
<td></td>
<td>3900</td>
</tr>
<tr>
<td>P instructions generated</td>
<td>23,500</td>
</tr>
<tr>
<td>CAL instructions generated</td>
<td>38,100</td>
</tr>
<tr>
<td>Size of code (54-bit words)</td>
<td>18,200</td>
</tr>
<tr>
<td>Compile, translate, assembly time (CPU sec.)</td>
<td>43</td>
</tr>
</tbody>
</table>

* this includes the run-time package

The compiler and translator run-time are currently dominated by the character by character I/O (about 50% of the total time). One of our current projects is directed toward improving the run-time support.

9. Reliability:
Most of the programs we have compiled were first debugged with PASCAL-6000, so error recovery hasn’t really been tested. The P-compiler has had quite a bit of use at Stanford. No errors in the generated code have been detected for several months, and we have compiled and executed John Banning’s 10,000 line PASCAL Analyzer program (PASCAL Newsletter #5).

10. Method of Development:
CRAY PASCAL was bootstrapped using PASCAL-P and PASCAL-6000. A total of 3 machines (CDC 6600, Cyber 73, 7600, Data General Eclipse, and the CRAY-1) and 3 character sets were involved in the bootstrap process. Approximately 6 man-months were required. Both implementors have previously modified batch OS/360–370 compilers to run interactively under ORYEL/370 (including CRAY-OS).

July 7, 1977

ALGOL-W, PL/C, and Sassan Hazeghi’s PASCAL-P compiler) and are experienced system programmers. Neither implementor had ever used a CDC 6600 or a CRAY-1, or written large PASCAL programs before the project started.

Subsequent development has been done using a PDP-11/70 running UNIX, with a link to the CRAY-1 for compilation and testing.


No libraries are currently available or planned. External procedures (declared as FORTRAN, though functionally requiring PASCAL calling conventions) are supported and have been used. Limited separate compilation is supported by allowing second level procedures (procedures declared in the PROGRAM block) to be entry points.

Data General Nova/Eclipse — Introduction.

Since the announcement in PDCN 8 of a Data General implementation by R. E. Berry, we’ve witnessed a lot of activity this summer. As an experiment, we are going to try to get everyone together here!

Thanks to Rodney Thayer, Central Research Group, P.O.Box 451, Harvard, MA 01451 (617/772-2306) who wrote 7/7/77: “a few people in my area (myself included) are investigating R. E. Berry’s U. of Lancaster PASCAL for the Data General NOVA. If I am closer than England for somebody, they are welcome to write to me to find out about Lancaster PASCAL."

On 7/8/72, Gregg Marshall at the National Oceanic and Atmospheric Administration in Denver, CO 80200 (303/499-1000 X4482) wrote out a checklist for the Lancaster NOVA PASCAL implementation, "in case they haven’t sent one, too." (They hadn’t.) Its information is included in the summary below.

Other NOVA implementations have been reported by Ted Park, A. J. Hurst, and Rafael Bonet — see below. H. S. Magnuski, Gamma Technology, 800 Welch Road, Palo Alto, CA 94304 (415/326-1661), wrote on 7/7/72 that he is trying to obtain some NOVA implementations for evaluation. Hopefully he will report his findings to PDCN. On 7/7/72, Bruce MacKenzie, Computervision Corp., 201 Burlington Road, Route 62, Bedford, MA 01730 (617/725-1800), announced that "We will be implementing Pascal on Data General's NOVA's and NOVA compatible machines in two weeks [i.e., at this conference]."

Also, Larry Walsh, KULM Corp., 4900 Old Iron Sides Drive, Santa Clara, CA 95050 (408/988-2900) is looking at Pascal-P for the KULM 1664, a ruggedized NOVA.
Requests for Data General implementation information have come from:

77/07/11: Doug Kaye, Computer Services, Du Art Film Labs, 235 West 55 St., New York, NY 10019 (212/757-4580).
77/07/14: Mike Tiller, 2501 N. Lancaster Lane #178, Plymouth, MN 55441 (612/546-6687).
77/08/10: Kevin Driscoll, 330 SE 11th Ave., Minneapolis, MN 55414 (612/331-2133).
77/07/14: Wayne Seipel, James Peterson, Computer Science Dept., University of Texas, Austin, TX 78712 (512/471-1773).

Data General ECLIPSE (Loma Linda)

LOMA LINDA UNIVERSITY

June 3, 1977

Dear Andy,

I thought this might be the first, but I see from the latest newsletter that at least one other Data General version exists.

However, I would like to report another Pascal P4 system solely designed for the Data General Eclipse Series computers with floating point hardware (since the Eclipse enhanced instruction set is heavily used, my Pascal will not run on a Nova).

I would be willing to disperse DG compatible dumps of the system to interested users who supply their own mag-tape. I am not in a position to supply documentation, so interested parties would still need to get the implementation kit from the University of Colorado.

To ease the implementation, I used a single size data unit -- 64-bits for everything. A virtual memory (paging) scheme is employed so that the system will run in almost any memory configuration.

The assembler for PCODE and interpreter are both written in DG assembly language. I am quite pleased with the speed of the system. It takes something over an hour to compile the compiler (~4000 lines of code). This is only 4 times slower than the vendor supplied FORTRAN compiler. (And the Pascal system is interpreted with software paging!!)

The specifications of the system are as follows:

- worksize = 64 bits
- memory size = 64K words
- integer size = 32 bits used in all calculations (64 bits stored)
- real size = 64 bits

I have implemented the entire interpreter except the transcendental functions and the I/O of 'real' data. The transcendental functions are, at present, of little interest so I may be several months before implementing these. I/O of 'real' data is needed. I am working on it and will have it ready in a couple of weeks.

I have read several comments in the PUG newsletter indicating how many people perceived the bootstrapping process as being rather difficult -- indeed, the implementation kit seemed to indicate this also. I would like to offer my encouragement to those who try by pointing out that the implementation kit we received from George Richmond at the University of Colorado was quite complete and bug free. I was able to have the compiler compile itself correctly after less than one man-month effort.

Sincerely,

-Andy Mickel

 Ted C. Park
 Technical Specialist
 New address: Ted C. Park
 Medical Data Consultants
 Suite 302
 1894 Commercenter West
 San Bernardino, CA 92408
 714/ 825-2683

The Scientific Computation Facility is a Biotechnology Research Resource supported in part by NIH grant RR 00276.
Data General NOVA 840 (Barcelona)

5 June 1977

Pascal User's Group

Dear Mr. Michael:

I have received Pascal Newsletters #5 and #7 on the same enclosure from the University of Southampton, Great Britain. What about #6?

My company, SEDINSA-TELESINCRO is a holding owned by the spanish government for the development of the national computer industry.

We bought CALTECH'S SOLO SYSTEM to experiment it as a software tool running in the DGC'S NOVA B40 at the research & development department. The NOVA is used as a software factory.

Enclosed is a short report about our implementation. Sorry but distribution is not planned.

I have personnel interest in PASCAL, so the address in the PUG mailing list is my home address. My office address is the implementors' address below.

Sincerely yours:

Rafael M. Bonet

cc: P. Brinch Hansen

1.- Implementors:
Rafael M. Bonet
Arsenio Lago
Ramón Cervelló
TELESINCRO S.A.
Departamento de Investigación y Desarrollo
Rocafort 100
Barcelona 15
SPAIN
Phone: (93) 3254100

2.- Machine:
Data General Corp., NOVA 840

3.- Operating System:
SOLO SYSTEM

Minimal Hardware Configuration:

CPU options: Floating Point Unit
Automatic Multiply/Divide Unit
Real Time Clock

Memory Map & Protection Unit (MMPU)

Disk: DIABLO model 33 (2.5MBytes)

AMPAC model DM448 with western Peripherals Interface

Tape: WANGCO 800 bpi, 9 tracks, 45 ips

Printer: TALLY 200 1pm with Data Products Interface

Card Reader: DOCUMENTATION 600 cpm with Documentation Interface

Console: Standard

Also supported by the system:
Second Console
4060 Multiplexer
as much memory as supported by the MMPU

4.- Method of distribution:
The SOLO SYSTEM and its distribution is not a company objective. Thus, we have no plans for distribution. But we shall study each request of a system copy.

5.- Documentation available:

Our system is an implementation of the CALTECH'S SOLO SYSTEM.
The languages description is given in two CALTECH Manuals:

- Concurrent PASCAL report.
- Sequential PASCAL report.

The system works in interpretive mode. The NOVA interpreter, an assembly program, is documented in spanish.

6.- Maintenance:
The high level coding (CPASCAL or SPASCAL) was written at the CALTECH by Per Brinch Hansen's team. Neither CALTECH nor Per Brinch Hansen (now at the Southern California University) provide maintenance for this software.
The low level coding (the NOVA interpreter) is responsibility of our team, but our structure does not allow a formal maintenance. Of course, we accept error reports.

7.- Standards:
CALTECH sequential PASCAL is not a standard PASCAL implementation as you can find in the CALTECH report.

8.- Compiler / Interpreter:
The system is interpretive. The only portion in target machine code is an assembly program, called the kernel, with a size of 5K words of 16 bits. The PASCAL interpreter, included in the kernel is 2 K words long. The SOLO runs sequent PASCAL programs. The compilers speed is about 90 char/sec.

9.- Reliability:
The kernel reliability is excellent.
For the compilers, some not important bugs were detected. Some of them were fixed. In general the reliability is good.
10. Development method:

   The tapes from CALTECH were used to implement a bootstrap SDLO SYSTEM running under DGCS's Real Time Disk Operating System. Then we developed our stand-alone SDLO SYSTEM.

   The system can produce a backup tape. This tape is loaded into disk by means of the IPL operation and an AUTOLOAD program written at the beginning of the tape.

   Once the tape on disk the system is loaded by IPL.

   For the people interested only in sequential PASCAL, it is possible to write an interpreter (or compiler) of sequential PASCAL, changing the SYSTEM CALL instruction from a branch to concurrent code to the actual execution of the function required. As Per Brinch Hansen says, it is a man month work, but it doesn't exist a documentation about how to do it.

Data General NOVA (Lancaster)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER, R. E. Berry and A. Foster; Dept. of Computer Studies; University of Lancaster; Bailrigg, Lancaster L11 4YX, U.K.; 65201 (STD 0524).

2. MACHINE. Data General Nova series (2/10, 820).

3. SYSTEM CONFIGURATION. RDOS 4.02/5.00 operating system; 32K core, disk backing store.

4. DISTRIBUTION. Cassette tape or 2.5 Mbyte cartridge disk.

5. DOCUMENTATION. A user manual is provided.

6. MAINTENANCE. No formal commitment to provide support; Release 2 under development and will subsequently be consolidating bug reports accepted on Release 1.

7. STANDARD. Pascal 64 subset accepted.

8. MEASUREMENTS. Typical runtimes compare favorably with those of other languages generally available on the Novas. P-code is generated, assembled and then interpreted. (* Compilation and execution space requirements not reported. *)

9. RELIABILITY. (* Thought to be good. Number of sites using system not reported. Date first released not reported. *)

10. DEVELOPMENT METHOD. Originally cross-compiled from a CDC 7600. The P-code assembler was written from scratch in Pascal; the P-code interpreter was implemented in Nova assembly language.

11. LIBRARY SUPPORT. (* No information provided. *)

DEC PDP-8 Minnesota

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER, John T. Easton, 612/373-7525; James F. Miner, 612/373-9916; Jonathon R. Gross, 612/833-4884; Address correspondence to: Pascal Group; SSRFC; 25 Blegen Hall; University of Minnesota; 269 19th Ave. South; Minneapolis, MN 55455; 612/373-5599.

2. MACHINE. Digital Equipment Corp. PDP-8/e

3. SYSTEM CONFIGURATION.

   OS/8 version 3. Hardware required:
   -K88-E (EAE with mode 8 instruction set)
   -K88-E disk, or other direct access mass storage device with at least 131K 12-bit words (e.g., RF32 or RF60).
   -16 K minimum of core/RAM. 32 K is highly recommended.

4. DISTRIBUTION. Not yet ready for release.


6. MAINTENANCE. A policy has not yet been determined.

7. STANDARD. Emphasis has been placed on close adherence to the Pascal User Manual and Report. There are two major restrictions: a) Parameters may not be procedures and functions. This restriction will not be lifted without full type checking (which requires a change in the Pascal Standard). b) Files may be declared only in the main program, and files may not be components of arrays, records, or files; nor may files be allocated with the procedure NEW. Minor restrictions: set size=96 elements; maxint=8,388,607 (2**23-1). Full ASCII character set is supported.

8. MEASUREMENTS.

   Execution speed—roughly comparable to FURTRAN IV (F4). I/O seems to be faster than FURTRAN, while computation seems slower.

   Execution space—Interpreter takes 12K, space needed for P-code and run-time storage depends on program.

   Compilation speed—much slower than F4. We hope to make some improvements in this area. About 30 characters/sec. presently (77/07/30).

   Compilation space—65K 12-bit words to compile itself.

9. RELIABILITY. Fair to good and improving. The system is has been in use at 1 site since 76/11.

10. DEVELOPMENT METHOD. As with most languages on the PDP-8, Pascal makes use of an interpreter (a modification of P-code) written in PALR. The compiler (about 5400 lines, based on Pascal-P4) and assembler are written in Pascal. All standard procedures are written in PALR. Because of the design of the system, the implementation is not suitable for real-time applications. On the other hand, the implementation does provide 131K words of virtual memory for code and store. Effort involved has been 1 person-year for applications programmers without previous experience writing compilers.

11. LIBRARY SUPPORT. Currently (77/07/30), none.

Digital Equipment Corporation (DEC) PDP-11 -- Introduction

At one time last year (PUG8 6-7) Steven Schwarz and C. E. Bridge at DuPont wrote to say they were coordinating a DECUS SIG Pascal. We thought they would coordinate PDP-11 implementations. Well, they haven't, and they have not been communicating either. We've heard that DECUS SIG Pascal is in other hands.

Interest in PDP-11 Pascal has been high. But from our point of view there are far too many Pascals on the 11 to wade through.

A few comments: Electro Scientific Industries Pascal for the 11 has received another good report — see the letter from Wayne Raskand. Structured Systems has come up with an implementation which runs on many operating systems including UNIX. The highest quality RSK-I system we've had reports on comes from Stockholm. Finally, we have news of UNIX Pascal from U.C. Berkeley.

Jim Shores, with the US Navy in Connecticut, phoned on 77/05/24 and reported he had a Brinch Hansen interpreter running as a task under RSK. Also he phoned Bob Lucas at NBS in Maryland and found out that Bob doesn't think too much of his own RSK implementation. With all the others around now, that's okay.

See also HERE AND THERE News section under David Miller, Matti Karinen, John Nunnally, Alfred J. Hubbert, Martin Tuori, and Aron Insinga.

--Andy Mickel
Dear Andy:

I suspect that readers of the PASCAL Newsletter may get the impression that there does not exist a reliable standard PASCAL compiler for the PDP-11 that is useful for production work, but from our experience this is simply not the case. We are using the compiler from Electro Scientific Industries (ESI) under the RT-11 operating system on four different PDP-11 systems (11/03, 11/04, 11/20, 11/40) for real-time laboratory applications and image processing. We have found ESI PASCAL better suited for process control type applications than the DEC FORTRAN. It generates in-line as opposed to threaded code. It allows direct access to I/O device registers as opposed to requiring subroutine calls. It provides a more efficient interrupt handling capability and allows insertion of assembler language statements in-line.

ESI PASCAL has also proven more practical for use on small PDP-11 configurations, such as a 1K 11103 with dual floppy drives, because it requires less memory and disk space. The ESI compiler (written in MACRO) is half the size of DEC's FORTRAN compiler and the PASCAL run-time support library is one-third the size of the FORTRAN library.

Sincerely,

Wayne Rasband
Section on Technical Development
National Institute of Mental Health
Bldg. 36, Rm. ZA-03
Bethesda, Md. 20014
301-496-4957

DEC PDP-11 (ESI)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. John Ankorn; David Rowland; Electro-Scientific Industries; 13900 NW Science Park Dr.; Portland, OR 97229; 503/641-4141; TELEX: 360273.

2. MACHINE. Any model Digital Equipment Corp. PDP-11.

3. SYSTEM CONFIGURATION. Minimum of 16K words. Operates under RT-11. Currently (76/11/02), an RMX-1M implementation is underway.

4. DISTRIBUTION. Compiler, support module, cross reference, text editor and instruction manual available for $1500. (* Tape formats, etc. not reported. *)

5. DOCUMENTATION. Over 70 page machine retrievable instruction manual. Currently (76/11/02) working on more.

6. MAINTENANCE. One year of unlimited fixes and updates, followed by annual subscription service. (* Reported by users that "vendor seems to be responsive in terms of support". *)

7. STANDARD. Full standard plus extensions: additional features for real-time hardware control; separate compilation of procedures; macro (assembler) code in line insertion; actual core addresses of variables can be fixed (giving access to external page 1/0 addresses at the Pascal level.

8. MEASUREMENTS. Compilation speed—About 3500 characters /second, on the PDP-11 model 05. Compilation space—very economical—it can compile 3000 line programs in 28K on PDP-11/40. No overlays are used in the system. Execution speed—about twice as fast as the DEC FORTRAN IV and many times faster than DEC BASIC. A worst-case 'number-cruncher' example ran at 401 faster than the DEC original FORTRAN. Execution space—very economical—much of the space improvement over DEC FORTRAN is due to the smaller support module for Pascal.

9. RELIABILITY. Excellent—far better than DEC FORTRAN. In use since 75/11. (* Number of sites using compiler not reported. *)

10. DEVELOPMENT METHOD. Single pass recursive descent compiler written in Macro-11. Hand-coded based on University of Illinois bootstrapping (with extensive changes) in about two person-years of effort. First compiler written by both implementors. Compiler translates source into Macro-11 which is then assembled and linked to the support module for execution.

11. LIBRARY SUPPORT. Separate compilation of procedures with load-time insertion and linkage is implemented.

DEC PDP-11 (Los Alamos)

---

Martin Rattner
STRUCTURED SYSTEMS CORPORATION
343 Second Street, Suite K
Los Alamos, California 87542
(505) 321-8111

STRUCTURED SYSTEMS CORPORATION is pleased to announce a new Pascal compiler for the DEC PDP-11. The STRUCTURED SYSTEMS Pascal compiler (PASCAL-SS) was designed and implemented by the team of A. Ian Stocks and Jayant Krishnaswamy, who previously developed the University of Illinois Pascal-11 compiler. The PASCAL-SS compiler is itself written in PASCAL and is self-compilable. It translates Pascal source programs directly into machine code. The language implementation is closely based on Jensen and Wirth's revised report (1975) with a number of language extensions and additional features aimed at large-scale system development in a production environment. Versions of PASCAL-SS are implemented or under development to run under the most popular PDP-11 operating systems, including DOS, RT-11, RSX-11 and UNIX.

Many features have been incorporated into STRUCTURED SYSTEMS PASCAL-SS which make it one of the most powerful and convenient-to-use Pascal systems for a production environment. Extensive compile- and run-time error checking and reporting features are incorporated in the compiler. Compiler $f$ options include an identifier cross-reference, automatic formatting/indentation of source listings, conditional compilation of sections of the source, a macro-expansion pass (similar to DEFINE in Burroughs Algo), and textual inclusion of library files in the source stream. Intensive object code optimization may be specified.

Programs and routines may be defined in separately compiled modules and linked together. User-controlled overlays permit very large programs to be compiled and executed under severe core constraints.

Anyone wishing additional information on PASCAL-SS should contact:

Martin Rattner
STRUCTURED SYSTEMS CORPORATION
343 Second Street, Suite K
Los Alamos, California 87542
(505) 321-8111
Dear Andy,

The newsletters are really interesting to read, although distribution is somewhat slow. When the May issue finally appeared in the end of July I found that I had mixed up my own address in the implementation note describing our PDP-11 compiler. I enclose an updated version.

As you can also see, I have decided to distribute the compiler myself. Dr. Schwar of nu Port have promised to distribute through DECUS, but I haven’t heard anything from him yet.

Truly yours,

Seved Forstendahl

1 IMPLEMENTOR

Seved Forstendahl
Address: Telephone 44 L' Ericsson
AUX/131 Stockholm, Sweden

2 MACHINE

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

The compiler generates code for floating point hardware and extended arithmetic if option switches are set.

3 OPERATING SYSTEM

MSX-11 or IAS, (DEC-10 crosscompiler under UPCs-10). Probably it is an easy task to replace the MSX interfacing routines with new ones interfacing USB or 41-11, but I do not plan to do that here. Maybe routines to interface with MSX-113 will be made.

4 DISTRIBUTION

The compilers are available at no cost if tapes are supplied. The distribution set contains source and object modules of the compilers and the runtime library, command files for compiler generation and maintenance user manual and compiler generation instructions.

The compiler will be distributed at no cost if tapes are supplied for one or more of the following choices:

- three DECTapes in PDP-11 BUS format (DECUS and PDP11 users)
- one V-track magnetic tape in UEL format (well users)
- one V-track magnetic tape in industry compatible format (users of UEL and other compilers)
- one V-track magnetic tape in BUS format (PDP11 users).

DOCUMENTATION

A user manual complementing the User Notes.

MAINTENANCE

No responsibility but if errors are found reports will be distributed to known users.

RESTRICTIONS AND EXTENSIONS

The compiler is a modification of the crosscompiler from dr. orn 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 RAX file system

The following list is mainly a copy from orn's contribution in Pascal newsletter #2.

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 to char/word)
- for boolean arrays (packing optional), one boolean/bit type is declared, the procedures pack and unpack are not implemented
- only local jumps are allowed
- a pair of procedures, mark and release, to allocate and reallocate dynamic storage.

The following extensions have been implemented:

- function results can be of nonclassical 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 introduced 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 local declaration with the procedure block replaced by "extern".
**SOURCES 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-11 Pascal compiler from Harmony compiles the source, when it then compiles itself the PDP-11 version is created.

The size of the compiler is 8K words 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 62 Kwords for code leaving 10 Kwords for data. This is enough to compile fairly large programs. However, the overlay scheme reduces the compiler slow about 25% lines / minute with 8K14-compilable disks, less with 4K15 disks. With RSX-11M V3 using FLKs and a 64K partition the speed is increased (-1/2 times).

**RELIABILITY**

Excellent. The compiler is now in heavy use at three sites and is used at four more. No errors have been found during the last two months.

**SESE OF DEVELOPMENT**

The crosscompiler for PDP-11 running on DEC-11 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 tuned it compiled itself and the compiler was thus transferred to PDP-11.

The implementation effort until now is about 6 manmonths. Maybe a new version which performs some optimizations will be developed later.

**6. DOCUMENTATION**

Machine retrievable documentation package.

**7. MAINTENANCE**

We intend to correct reported errors for the next few years. Error reports and updates will be sent at irregular intervals to all those who have received the compiler, unless otherwise requested.
The concept is clear and easy to understand. This enables you to modify or extend the compiler for your special purposes.

The concept of a "virtual machine" makes you almost independent from machine and Operating System.

The complete interface to the Operating System is contained in a simple program prefix. This seems to be the greatest advantage.

The few restrictions to "Standard Pascal" did not matter that much to us.

ad 3.) One of the most difficult tasks was to design a suitable set of prefix routines (as an interface to RSA instead of the Solo Operating System of P.B. Hansen). These prefix routines are system functions that manage for example reading, writing and overlay loading.

Our main principle of design was to be as simple and clear as possible so that
- the programmer can learn to use the new interface as quick as possible.
- even in extreme cases it is obvious what happens
- I/O is efficient (time and core/discharge).

To put this concept through requires a lot of courage, for the users often want an I/O-system as complex as they are used to from other programming languages (FORTRAN). Moreover, RSX has a very sophisticated filesystem and it is hard to implement it in the PASCAL-system and not to use all the complex functions it contains.

As an example, look at the way files are handled by the new prefix. Only two types of files are supported: sequential textfiles and random access files with a fixed recordlength of 512 bytes.

There are three groups of prefix routines for the file handling:

1.) routines for file definition:

   PROCEDURE PAGFILE(u: UNIT, f:FILENAME);
   PROCEDURE TXFILE(u: UNIT, f:FILENAME);
   with the type definition

   CONST FILENAMELENGTH = 30;
   TYPE FILENAME = ARRAY [1..FILENAMELENGTH] OF CHAR;
   CONST MAXUNIT = 4;
   TYPE UNIT = 1..MAXUNIT;

   These routines associate a page- or textfile with an unit number.

2.) file management routines:

   PROCEDURE CREATE(u: UNIT; INITIALSIZE: INTEGER; c: CONTIGUOUS);
   PROCEDURE CREATETEMP(u: UNIT; INITIALSIZE: INTEGER; c: CONTIGUOUS);
   PROCEDURE OPEN(u:UNIT; ACCESS: FILEACCESS);
   PROCEDURE CLOSE(u: UNIT);
   PROCEDURE DELETE(u: UNIT);

   "Create" and "create temporary" create a new file and "open" opens an existing file.

   "contigoustype" and "fileaccess" define the method of allocation and access.

   TYPE FILEACCESS = (READONLY, MODIFY, EXTEND, APPEND, READSHARE);
   TYPE CONTIGUOUS = (NONCONTIGUOUS, CONTIGUOUS);

3.) routines for reading and writing

   CONST PAGELENGTH = 512;
   TYPE PAGE = ARRAY [1..PAGELENGTH] OF CHAR;
   CONST LINESIZE = 122;
   TYPE LINE = ARRAY [1..LINESIZE] OF CHAR;
   PROCEDURE READPAGE(u: UNIT; n: INTEGER; var block: PAGE);
   PROCEDURE WRITEPAGE(u: UNIT; n: INTEGER; var block: PAGE);
   PROCEDURE READCHAR(u: UNIT; var c: CHAR);
   PROCEDURE WRITECHAR(u: UNIT; c: CHAR);
   PROCEDURE READLINE(u: UNIT; var text: UNIV LINE);
   "readpage" and "writepage" are for random access pagefiles.
   The other routines are for sequential textfiles. Instead of the type "page" any other type with the same length can be used.

   We don't claim to have invented new functions. On the contrary we have omitted as much as possible from the RSX-file system options without restricting its feasibility for the PASCAL user.

   But how to work with those simple I/O routines?

   A good practice will be the following one:

   The programmer chooses a set of "I/O-operators" for his special needs. These operators are procedures and functions written in Pascal. The programmer takes them out of a programlibrary or writes them himself or modifies existing programs for his purposes.

   An example for such a set will be:

   procedure readinteger (var n: integer, length: integer);
   procedure writeinteger (ni: integer, length: integer);
   procedure skipdelimiter; procedure newline;
   and so on

   The procedure readinteger does what you expect: It reads an integer "n" with at most "length" characters from the inputstream ending with the next delimiter.

   The only systemroutine used is "read one character from an inputstream". In Pascal procedures like readinteger can easily be written. If the programmer is in doubt what the program really does, one glance at the pascal source program (instead of considering twenty rules in a manual) obviously will explain it.

   This method is the best one to meet the need for structured, modular, portable and flexible programs.

ad 4.) A Pascal version for easily programming CAMAC Systems is under work and will be running summer 1977.

The implementation of Concurrent Pascal in the Operating System RSX11M using the task synchronisation facilities of RSX11M will be completed at the end of the year.

Afterwards we are planning to use (Concurrent) Pascal and the conditional critical regions-concept for multiprocessing applications (with microprocessors Intel 8080).

If you are Interested in our work, please write to us.

Sincerely yours,

Dipl.-Ing. K. Mayer
DEC PDP-ll (Belgium)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Pierre Verbaeten; K. V. Leuren; Applied Mathematics and Programming Division; Celestijnenlaan 208, B-3030; Heverlee, Belgium; (* No phone number provided. *)
2. MACHINE. Digital Equipment Corp. PDP-ll.
3. SYSTEM CONFIGURATION. UNIX. (* Minimum hardware requirements not reported. *)
4. DISTRIBUTION. (* No information provided. *)
5. DOCUMENTATION. (* No information provided. *)
6. MAINTENANCE. (* No information provided. *)
7. STANDARD. (* No information provided. *)
8. MEASUREMENTS. (* No information provided. *)
9. RELIABILITY. (* No information provided. *)
10. DEVELOPMENT METHOD. (* No information provided. *)
11. LIBRARY SUPPORT. (* No information provided. *)

DEC PDP-ll (Portland)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Barry Smith, Oregon Museum of Science and Technology, Computing Department, 4015 SW Canyon Road, Portland, OR 97221 (503/248-5923).
2. MACHINE. Digital Equipment Corp. PDP-ll.
3. SYSTEM CONFIGURATION. RSX/E. (* Minimum hardware requirements not reported. *)
4. DISTRIBUTION. (* No information provided. *)
5. DOCUMENTATION. (* No information provided. *)
6. MAINTENANCE. (* No information provided. *)
7. STANDARD. (* No information provided. *)
8. MEASUREMENTS. (* No information provided. *)
9. RELIABILITY. (* No information provided. *)
10. DEVELOPMENT METHOD. (* No information provided. *)
11. LIBRARY SUPPORT. (* No information provided. *)

DEC PDP-ll (Berkeley)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Charles Haley, William Joy, and Ken Thompson, Computer Center, Evans Hall, University of California - Berkeley, Berkeley, CA 94720. (* No phone number provided. *)
2. MACHINE. Digital Equipment Corp. PDP-ll.
3. SYSTEM CONFIGURATION. UNIX. (* Minimum hardware configuration not reported. *)
4. DISTRIBUTION. (* No information provided. *)
5. DOCUMENTATION. (* No information provided. *)
6. MAINTENANCE. (* No information provided. *)
7. STANDARD. (* No information provided. *)
8. MEASUREMENTS. (* No information provided. *)
9. RELIABILITY. (* No information provided. *)
10. DEVELOPMENT METHOD. (* No information provided. *)
11. LIBRARY SUPPORT. (* No information provided. *)

DEC PDP-ll (PAR)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Michael N Condict; PAR Corporation; On The Mall; Rome, NY 13440; 315/336-8400.
2. MACHINE. Digital Equipment Corp. PDP-ll/45.
3. SYSTEM CONFIGURATION. RSX-11d. Minimum hardware same as for RSX.
4. DISTRIBUTION. None until at least 77/06.
5. DOCUMENTATION. None yet. (* Not known if documentation will be machine retrievable. *)
6. MAINTENANCE. None yet.
7. STANDARD. Full Standard, probably with extensions.
8. MEASUREMENTS. Expected to be about 5000 FORTRAN source lines and 3000 Pascal source lines. Expected to run rings around FORTRAN compiler. (* Rich Cichelli reports on 77/08/31 that it is a 2 pass system in which the code generated is faster than the 19 (!) pass optimizer for William Wulf PDP-ll Fortran! *)
9. RELIABILITY. Will not be distributed until it is.
10. DEVELOPMENT METHOD. One pass Pascal to FORTRAN translator. Initial version of each procedure written in Pascal and then hand translated into FORTRAN. When compiler is finished or can compile itself it will be restored to its original Pascal in a massive inverse translation, and then run through itself, thus completing the bootstrap. Currently (76/12/14) project has consumed about 4 person-months. Expected to consume 6 to 9 person-months in all (with 1 person devoting half-time). Implementor previously built a compiler for a subset of Pascal for a class project, but has never written any program this large before.
11. LIBRARY SUPPORT. (* No information provided. *)
DEC LS1-11 (San Diego)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Pascal Group; Institute for Information Systems; UCSB Mailcode C-021; La Jolla, CA 92093; (* No phone number reported. *)

2. MACHINE. Digital Equipment Corp. LS1-11 Microprocessor, PDP-11 any model, TRAM 8510 and 8510A.

3. SYSTEM CONFIGURATION. Comes with a one-user operating system. Apparently requires some mass storage (disk or floppy disk).

4. DISTRIBUTION. Distributed on floppy disk in two versions: 1) Complete release: including all source code and internal documentation ($200); and 2) Code release: including system code and users manual ($50).

5. DOCUMENTATION. For complete release: compiled listings of all source code, and user and system maintenance documentation as complete as it exists. For code release: Users manual but no detailed system documentation. Documentation is machine retrievable.

6. MAINTENANCE. For complete release: compiler updates at least 3 times during 77/8/1 thru 78/8/1. For code release: No continued support for later releases. Only minimal assistance in response to telephone inquiries. Future plans: plan to have a version of this system for the Zilog Z-80 ready. Plan to have versions for Intel 8080a ready by 77/9, MOS Technology 6502, and Motorola 6800 ready by summer of 1978.

7. STANDARD. Pascal-P subset plus strings.

8. MEASUREMENTS. 700 lines per minute compile speed. 20K byte compiler, 10K bytes for resident monitor, interpreter, and run-time support.

9. RELIABILITY. Reported good. First released on 77/8/1.

10. DEVELOPMENT METHOD. (* No information reported. *)

11. LIBRARY SUPPORT. Extensive graphics software, text editor, text formatter, pretty printer, all in Pascal.

Hardware requirements are:

PDP-11/20 or up.
28K words of addressable core store.
either a DEC RF-11 or 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.)
a DECtape unit (we can supply the system only on DECtapes).

4. DISTRIBUTION. 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.
4) Binary for our operating system.

If you are interested in obtaining this software, please send the following to the distributor:

1) Three DECtapes (these must be in PDP-11 format).
2) A statement of your intended uses.*
3) One signed copy of Prof. Synder's enclosed letter.*
4) A stamped, self addressed mailer for returning your DECtapes (total weight is about 900g (2 pounds)).

*The Pascal-11 compiler was developed at the University of Illinois - Urbana and is copyrighted by its Board of Trustees. The work was supported in part by a grant from the National Science Foundation. 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.

5. DOCUMENTATION. Unfortunately, very sparse at present (77/01/21) but we shall include in the distribution package all that is available. (* This is apparently not machine retrievable. *)

6. MAINTENANCE. 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 another system.

7. STANDARD. Differences:

"With" unimplemented.
types real and set unimplemented.
variant records not permitted.
procedures-as-parameters not permitted.
write, readln not implemented
EOL feature still included.
Extensions:
compile time options.
source level library routines.
overlays.

8. MEASUREMENTS.

compilation speed--(* No information reported. *)
compilation space--(* No information reported. *)
execution speed--(* No information reported. *)
execution space--(* No information reported. *)

9. RELIABILITY. (* Information on reliability not reported. Number of sites using system not reported. Date first released not reported. *)

10. DEVELOPMENT METHOD. (* No information provided. *)

11. LIBRARY SUPPORT. Source level library routines are implemented.
Dear Mr. Michel,

as I have indicated by a letter mailed on February 14, 1977 our DECSystem-10 Pascal compiler of December 30, 1976 is now distributed by DECUS. Mr. Nigel Derrett from Aarhus/Denmark pointed out one error in our PASCAL implementation of December 30, 1976: The attempt to pack a variable of a subrange type that requires exactly 35 bits - one less than an entire word - may result in an infinite loop.

Another, although minor, bug is connected with reading from TTY: in order to avoid unnecessary prompting of input during opening input from the TTY, the compiler checks whether any reading from TTY is requested during a program. The asterisk - prompting input to fill the first TTY-buffer - will only appear, if input from the TTY will be requested somewhere in the program. Unfortunately, arguments of standard procedures have not been included in this test. Therefore, if input from TTY appears within a program only as first argument to GETFILENAME, the input device TTY will not be opened automatically. An easy way around this weak point consists in inclusion of e.g., a statement HEADLN(TTY).

Both errors will be removed in the next compiler version which, however, may take some time. I would like to investigate means to further optimize code generation by e.g. improving the allocation and use of registers. Since any change at such a sensitive area has to be made very carefully, it will take some time.

A PASCAL cross compiler running on the DECSystem-10 and generating code for a German minicomputer Diets MINCAL-621 is currently being converted to software paging of procedures: the pure code of procedure bodies is allocated in 128 Byte pages that may be loaded from disk to a certain core area and may be overwritten if that core area is needed. The nesting of 131 simple procedures has been successfully tested to verify the loading, overwriting and reloading of procedure bodies into core. Next we want to implement the PASCAL system (which is already available by a non-paging cross compiler for this MINCAL-621) by this new software paging PASCAL system and to compare its performance with paging versus the one without paging.

A compiler for Concurrent PASCAL has been developed by a group of students at our laboratory in collaboration with H. Kemen and myself. This is an implementation completely independent from that of Brinch-Hansen for the PDP-11/45. Our Concurrent Pascal compiler is executed as a Pascal program on the DECSystem-10 and generates code for a hypothetical intermediate machine which has been designed to facilitate easy code generation for Byte-oriented minicomputers. Two code generators have been implemented, one for the MINCAL-621 and one for the INTERDATA M95. Using this Concurrent Pascal implementation an assembler program to control our TV-periphery connected to the MINCAL-621 has been reimplemented as a system of Concurrent Pascal processes. The ease of designing a process system for actual applications in Concurrent Pascal has encouraged us to proceed with the idea to program our local inhomogeneous computer network (two different MINCAL-621, a PDP-11/20 and a PDP-10) in Concurrent Pascal. A code generator for the PDP-10 has just been completed and is currently being tested. In the course of writing a code generator for the PDP-10 (36 bits per word) we realized some of the shortcomings in the definition of the intermediate hypothetical machine which was originally conceived for byte-oriented machines.

Nevertheless, we have already executed a system of Concurrent Pascal processes on the PDP-10 and another one which communicated from one MINCAL-621 to another.

Our Concurrent Pascal Compiler is described in a report (in German):

CONCURRENT PASCAL Compiler für Kleinrechner
B. Brügge, B. Bisch, Th. Kahl, H. Linde, M. Mittelstein, H. Westphal
IfI-HH-0-30/76 (December 1976)

Sincerely yours,

DECE-10 (Hamburg-DECUS)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Implementor/Maintainer: E. Kisicki; H.-H. Nagel; Universität Hamburg; Institut für Informatik; Schütterstraße 66-72; D-2000 Hamburg 13, Germany; 040-4133-151; TELELEX: 314731 und 56d. Distributor: DECUS; Maynard, MA 01754; USA: 617 897-5111; TELELEX: 94 8457; TWX: 710 347 0212.

2. MACHINE. Digital Equipment Corp., DEC-10. (Adapted to the DEC-20 by DEC).

3. SYSTEM CONFIGURATION. DEC TOPS-10 monitor using Concise Command Language (CCL). Uses K=10 instruction set. Modifications to use K=10 instruction set have been made by Charles Hedrick.

4. DISTRIBUTION. DECUS (Digital Equipment Corp., User's Society) Maynard MA 01754 (617/897-5111; TWX: 710-3470212; TELELEX 948457). Also DECUS Europe, P.O. Box 340, CH-1211 Geneva 26, Switzerland (022) 42 79 50; TELELEX 22593).

5. DOCUMENTATION. Machine retrievable manual included on distribution tape.

6. MAINTENANCE. No regular maintainance can be given.

7. STANDARD. Extensions: Functions FIRST and LAST for arrays; UPPERBOUND and LOWERBOUND for arrays; MIN and MAX available as standard functions; procedures to determine the value of CCL options available; "OTHERS" in case statement; LOOF...EXIT IF...ENO statement; Initialization procedure.

8. MEASUREMENT. (* No information provided. *)


10. DEVELOPMENT METHOD. Pascal-F2 and subsequent self bootstraps. Latest version dated 76/12/30.

11. LIBRARY SUPPORT. Symbolic post-mortem dump available. Interactive run-time source-level debugging package available. Separate compilation and inclusion in relocatable object code library of Pascal, FORTRAN, COBOL, ALGOL, and MACRO-10 assembler routines.

---
DEC-10 System Programmers' PASCAL - an alternative PASCAL system for users who need full access to the facilities of TOPS-10 or who want to do data processing.

(1) Charles L. Hedrick, Computer Science Dept., Rutgers University, New Brunswick, N.J. 08903.

(2) PDP-10 and KL-10 CPUs only. Probably PDP-20 with minor changes.

(3) TOPS-10 operating system. Virtual memory 601 or later monitor required. One minor feature requires 6.02.

(4) The latest stable version is distributed through DECUS. The most recent uncorrected version can be obtained from me directly (at the address above) if you send a blank master tape and return postage.

(5) A supplement to the Revised Report is included in the distribution in machine-readable form.

(6) I am currently maintaining it and will continue to do so for the foreseeable future, but I probably will not do further development work (i.e., adding features). I hope this version will be superseded by an improved version from Hamburs.

(7) GO TO out of the current procedure is not supported. (Tricky to implement, and a terrible idea anyway.) Local files not implemented. (No ECS on a DEC-10 and simulation with random-number named files seems unattractive.)

(8) Compiler plus interactive debugging package (PASDDT) and a library of useful system functions. Completely integrated into CCL (COMPIL).

(9) The compiler is quite reliable. The runtimes are reliable for standard PASCAL and the most commonly-used extensions. Some obscure corners of the extensions have not been well tested (mostly those involving user error recovery).

(10) Modified version of the Hamburg (Nagel) compiler. The latter was done from some edition of PASCAL-P. I believe, in several stages, I did not start out intending to have my own version of PASCAL. Rather I wanted to test out a few ideas, with the hope that Prof. Nagel would adopt those that turned out to work, for use in the original DECsystem-10 PASCAL (which he maintains) and which is available from DECUS. I believe this will happen in the long run, but in the meantime I am publishing my experiences in the hope that it will help other PASCAL implementors who are confronted with similar problems. Thus this note is directed at fellow implementors and is not intended as an advertisement for our version. (Indeed I recommend highly that other people use the Hamburg version unless they absolutely require some of our features.) In the following, an asterisk (*) indicates a feature not in the edition we submitted to DECUS. A plus sign (+) indicates a feature present in the Hamburg version.

(1) INITPROCEDURE(*): These specify initial values for variables. They do not compile code, but put the values in the initial core image directly. INITPROCEDURE BEGIN <assignment statements> END.

(2) OTHERS in case statement(*): OTHERS: <statement> will catch any cases not fitting anywhere else.

(3) LOOP(+): Allows a loop with exactly one exit in the middle. Note that this is still a one-in-one-out construct. LOOP <statements> EXIT IF <Boolean expression> <statements> END.

(4) Program statement(*). There was some question what the PROGRAM statement should do in interactive implementations. I believe Hamburger's solution is a good one. If any files are listed in the PROGRAM statement, the program begins with a dialogue asking for specifications for them. It is important that this dialogue can be suppressed by not listing any files. This gives the program the option of setting the file names in some other way and specifying them to the RESET or REWRITE directly. This follows my BASIC DESIGN PRINCIPLE: It should be possible to write a program that cannot be identified by its users as a PASCAL program. I.e., one should be able to take over error handling and file specifications if desired.

(3) Interactive files: RESET does an implicit GET in official PASCAL. This causes PASCAL programs to try to read from the user's terminal before starting the program when it is INPUT. That makes it impossible for the program to output a prompting message first or to write a program that doesn't have terminal input at all. Most implementations on interactive systems allow one to specify a file as interactive. Then when it is RESET, no GET is done. Instead the buffer is filled with null (in our case) or blank (for the CYBER, which doesn't have a null) and EDLN is set. This ability is also useful for mistakes where one might wish to issue a positioning command (space forward, rewind, etc.) before doing the first GET. The CYBER specifies files as interactive by putting a slash (/) after their name in the PROGRAM statement. We make this an option in the RESET statement. (See below.) Putting it in the RESET statement is helpful since not all files are listed in our PROGRAM statement. In our implementation, the user's terminal is the special file TTY, and is always interactive.

(6) End of line: The Revised Report seems to require us to set the buffer to blank at end of line. Also, the DEC-10 has several line-terminator characters. Thus one cannot tell which one has occurred. We put the actual terminator in the buffer. A blank seems useful for systems that have no line-terminator characters (e.g., CYBER). Programs that use IF EDLN THEN REWIND will work either way. (Our READLN also skips the line feed if the initial terminator is carriage-return.) Note that it requires two GETs to skip a carriage-return, inserted sequence, although a single READY will do.
(7) End of file: In PASCAL EOF is normally false for input and true for output. This lack of symmetry complicates the I/O runtime, and also confused users, and makes the implementation of update mode difficult. In update mode one can do both GET and PUT on the same file. However, if EOF is true, GET will give an error, and if EOF is false PUT will give an error. We have not really solved this problem. Fortunately under most circumstances PUTX is used rather than PUT in update mode.

(8) I/O to strings: STRSET(filenamearray, start(n), end(n)) allows regular input to be done from the array starting at element start and going to the end of the array. STRWRITE for output. This draws text from integer and visa versa using the standard READ and WRITE. It also provides a sort of four man's simulation of files in EOF. I am not enthusiastic about it, however, and would be happy to see it go away. One must require that the array be declared at the same lexical level as the file, so that all of the array is in the same place, and visa versa.

(9) READ applied to strings: READ(filenamearray, start(n), end(n)) will read characters into the array until the next end of line. This is really a very useful for conducting dialogues with the user. The alternative seems to be to require string quotes to delimit the string and do something like READ(filenamearray, set(n)), where the user specifies a set of break characters. The idea of string quotes is very lacking, and impedes the construction of simple dialogues. The break set idea is a good one, that we just have forgotten to use. If a variable is specified after the colon, it is set to the number of characters read. If more characters are typed than the size of the array the extras are ignored (but counted in length, so the program can tell what has happened). If too few are typed the rest of the array is filled with blanks.

(10) RESET and WRITE: Our RESET and WRITE are RESET(filename, filespec, implementation-dependent stuff) and WRITE(filename, filespec, implementation-dependent stuff). Filename is a string (PACKED ARRAY OF CHAR) of any length including a literal in string quotes. It contains a device/file specification in the standard DEC-10 format. Interactive is true to suppress the implicit GET as described above. The implementation-dependent stuff allows the user to control protection, version number, date, I/O mode, buffering, etc. In particular, it allows him to specify buffered or unbuffered I/O (or #), and to declare that the file is blocked (in the COBOL sense) (*). It also allows him to suppress the normal runtime error messages. ERSTAFILE can then be used to see what errors, if any, occurred.

(11) Variable size records(*): One can declare a file to consist of a type that involves variants. Then one can use GET(filename, variant, ...size), and similarly for PUT. As with NEW, this causes only the appropriate number of words to be read or written. Size is used when the last element in the declaration is an array. It specifies that only the first <size> elements are to be used.

(12) UPDATE(*): UPDATE is like RESET, except that it allows records to be updated in place. To use this read the record into the PASCAL buffer with GET. The contents of the buffer can then be revised. PUTX(filename) causes then new contents of the buffer to replace the original record in the file. Exactly the same number of bytes is chanced as was read by the original GET. No surrounding records are affected, no matter what sort of blocking is involved (except for unbuffered I/O). PUT may also be used in update mode when unbuffered I/O is being done.

(13) Random access I/O(*): On the DEC-10 random access devices have fixed size physical blocks. No standard "access methods" are provided. (Basically we have only BSAM and BSAM with format U records.) Thus one can either simulate the COROL runtime system, or stick with low level primitives. The latter seems consistent with PASCAL's philosophy. It has the disadvantage of being machine-dependent; however, since it depends upon the architecture of the DEC file system. (This architecture is probably quite common outside of IBM however.) We use an index into the file. SETPOS(file, index, suppress GET) sets one to the index'th byte in the file. (NB: byte not disk block. We perform the buffer manipulation needed to get to the specified byte.) CORPOS(file) returns the position at the beginning of the last record read or written.

(14) APPEND: APPEND is like WRITE, but begins writing at the end of an existing file.

(15) BREAK: BREAKIN: BREAK forces output buffer in buffered modes. BREAK clears the buffer in non-buffered modes. They are used before or after statement positioning, etc. BREAK would be needed to force output to the terminal except that the file TTY is handled with a special kind of I/O that does not use buffers.

(16) RENAME: RENAME(filename, filespec, implementation-dependent) renames the file open on <file>.

(17) CLOSE(file) closes the file. This is needed to make the file safe in case the system crashes. It also releases the I/0 channel for use by other files. (There is a maximum of 16 I/0 channels on the DEC-10.) All files are automatically closed at the end of the program.

(18) DISMISS(file) aborts creation of an output file.

(19) UPCODE(filename, Boolean expression) controls mapping of lower case to upper case.

(20) Routines are available to handle interrupts (via PSI), do interjob communication (IPCG), and call the standard DEC-10 command manager (BSAM/WILK).

Diets MICALS 621 (Hamburg)

See the letter from H.-H. Nagel under DEC-10.

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER: H.-H. Nagel; Universitat Hamburg; Schulerstrasse 60-72; D-2000 Hamburg 13, Germany; 040-4123451; TELEX: 214 732 uni hh d.

2. MACHINE: DEC-10 cross compiler producing code for the Diets MICALS 621 minicomputer.

3. SYSTEM CONFIGURATION. (* No information provided. *)

4. DISTRIBUTION. (* No information provided. *)

5. DOCUMENTATION. (* No information provided. *)

6. MAINTENANCE. (* No information provided. *) Currently being converted to software paging of procedures.

7. STANDARD. (* No information provided. *)
8. MEASUREMENTS. The pure code of procedure bodies is allocated in 128 byte pages that may be loaded from disk to a certain core area and may be overwritten if that core area is needed. (* No information provided. *)

9. RELIABILITY. (* No information provided. *) The nesting of 131 simple procedures has been successfully tested to verify the loading, overwriting and reloading of procedure bodies into core.

10. DEVELOPMENT METHODS. Cross compiler on the DEC-10 that generates code for the Dietsz MINCAL 621 minicomputer.

11. LIBRARY SUPPORT. (* No information provided. *)

Fujitsu FACOM 230

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Masato Takeichi, formerly at Department of Math. Engineering and Instr. Physics, University of Tokyo, Hongo-ku, Tokyo 113, Japan. Present address: Department of Computer Science, University of Electro-Communications, 1-5-1 Chofugakou, Chofu-shi Tokyo 182, Japan.

2. MACHINE. FACOM 230-38, 224K bytes.

3. SYSTEM CONFIGURATION. OS/2/VS. (* Minimum hardware required not reported. *)

4. DISTRIBUTION. (* No information provided. *)

5. DOCUMENTATION. See the article "Pascal Implementation and Experience", by Masato Takeichi, Journal of the Faculty of Engineering, University of Tokyo 34:1 pp 129-136.

6. MAINTENANCE. (* No information provided. *)

7. STANDARD. Restrictions: No local file variables; no parametric procedures.

8. MEASUREMENTS. Self compiles in 309 sec. Compiler object code is 117K bytes, monitor is 8K bytes, and self compilation requires 43K bytes of data store. Execution times, relative to Fortran, are given in the following table.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Self</th>
<th>Fortran</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix multiply</td>
<td>1.35</td>
<td></td>
</tr>
<tr>
<td>Sort</td>
<td>1.24</td>
<td></td>
</tr>
<tr>
<td>Additive partition</td>
<td>1.96</td>
<td></td>
</tr>
<tr>
<td>Character count</td>
<td>0.63</td>
<td></td>
</tr>
</tbody>
</table>

9. RELIABILITY. Working very well. (* Number of sites and first date of release not reported. *)

10. DEVELOPMENT METHODS. Based on H. Nagell's Trunk compiler (5800 lines of Pascal), with a Pascal monitor written in FASP. The initial version began working in October, 1975, after 2-3 months of work.

11. LIBRARY SUPPORT. (* No information provided. *)

Harris/4

From O. W. van Wijk (87/08/15+)

20EJBC, P.O. BOX 47, DELFT, HOLLAND. TEL 015-138222 TELEX 33567

THREE SHOT PASCAL P4 SYSTEM ON A HARRIS/4 COMPUTER.

AT THE jBEC WE HAVE A HARRIS/4 MACHINE WITH 64K OF 24 BITS WORDS.

The implementation was done as a student task by O.W. van Wijk from Delft University of Technology, department of mathematics. Starting with an implementation kit obtained from E.T.H.-Zurich it took about 600 hours to get a running version of the compiler that could compile itself.

The Pascal system then consisted of the p4-compiler, an assembler for the pascal instruction, written in fortran, and a runtime system of about 6K Harris assembler code. The fortran p4-compiler scans the source two times and generates during the second phase linked pages of subroutines calls.

One page = 1024 words code + 2 words link, these pages are stored on disk. Running a Pascal program is done by loading the runtime system, which is a normal background program. This program has ten parameters, one to specify the disk file containing the pages with the program object, the second to specify the number of pages held in core. The pages are allocated in the page area and swapped according the "least recently used" algorithm. In this construction the code size of a Pascal program is not limited by the size of the machine core, and allows the use of maximum 37K of the available 62K core, for Pascal data on stack and heap. Running Pascal this way with 15 pages in core, it takes the compiler about 6 min. to compile itself.

The second stage of the implementation was making some extensions to the compiler and small changes in the code generation. Extensions were made to allow the use of external procedures of different kinds of source languages. (Pascal, fortran, Harris assembler.) External procedures are declared in a prefix to the Pascal program. The type of parameters of external procedures is restricted to the standard types. Software to execute external was developed. External procedures of Pascal source are stored on disk in a form and included by the p4-assembler, now also written in Pascal, on the page file. External procedures of fortran/assembler source are stored on disk as relocatable modules, included by the p4-assembler as one record on the page file, loaded and relocated by
the runtime system. The use of external procedures also allows a kind of fortran-like direct access I/O, which was, among the use of existing programs, the main reason to make this extension.

core layout scheme for running Pascal

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STACK</td>
<td>Stack area for temporary data</td>
</tr>
<tr>
<td>HEAP</td>
<td>Allocation area for variables</td>
</tr>
<tr>
<td>P4 COMPILER</td>
<td>Front-end compiler for the P4 system</td>
</tr>
<tr>
<td>P4 ASSEMBLER</td>
<td>Front-end assembler for the P4 system</td>
</tr>
<tr>
<td>HP ALGOL</td>
<td>Library routines for HP-Algol</td>
</tr>
<tr>
<td>HP ASSEMBLER</td>
<td>Assembly routine for HP-Algol</td>
</tr>
<tr>
<td>HP COMPILER</td>
<td>Compiler for HP-Algol</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>Output area for compiled code</td>
</tr>
</tbody>
</table>

Heathkit H-11

(* This machine is based on the LSI-11 microprocessor from DEC and it is believed that the DEC LSI-11 (San Diego) implementation will run on this machine; though nothing definite has been reported. *)

Hewlett Packard HP-21MX (Durban)

See also HERE AND THERE News section under Tao-Yang Hsieh.

UNIVERSITY OF DURBAN-WESTVILLE

We bought the P4 system from Zurich early this year and after a few hassles with block sizes, end of files and character sets, managed to get the files to tape and also listed. Since then I have been attacking the problem on two roughly parallel fronts, namely implementation of the PASCAL defined by the P4 system on the two machines mentioned.

My major effort has been on the HP as I have easier access to it. I have rewritten the P4 interpreter in (of all things) FORTRAN chiefly because I could make use of its horrible features, such as EQUIVALENCE REALS and INTEGERS for the stack and so on. This now appears to be working and I have run a few hand compiled programs through it. However memory size limited the amount of code I could give the CODE and STACK arrays. This is alright for running small programs but the compiler itself would not fit. I have thus taken that interpreter and split it into two phases - a load phase and a run phase. The load phase now does two passes over the P4 code to produce the internal form of code on a disc file, then an array. The run phase is then a stripped down version of the normal interpreter with all irrelevant detail (post mortem dumps, trig functions etc.) eliminated. This will have a bigish CODE array and will basically operate on a virtual storage concept. This is still in FORTRAN but I will rewrite it in HP assembly soon. Thus as soon as I can get a compiler for the HP compiled I should be able to compile programs, but I feel that recompiling the compiler will be beyond me.

Now for the Univac. I am modifying the original FORTRAN interpreter to make allowances for the difference in architecture, etc. and will move that on to the Univac soon. Then by the usual bootstrap operation I will get PASCAL up there. That done I will probably bootstrap a more effective (non ininterpretable) system probably the BELFAST compiler for the 1900 series.

Thus by the end of July I should be able to compile programs (albeit slowly) on both machines and by the end of the year have efficient systems going on each. Next year all my students will learn PASCAL as a first language as a matter of course in their algorithms and problem solving course.

Hewlett Packard HP-2100 (Trieste, Italy)

---

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Mattia Hmeljak; Instituto di Electrotecnica ed Elettronica; Universita di Trieste; Trieste, Italy; Tel. 040-733033.
2. MACHINE. Hewlett Packard HP-2100.
3. SYSTEM CONFIGURATION. (* No information provided. *)
4. DISTRIBUTION. (* Unknown, implementation not yet complete. *)
5. DOCUMENTATION. (* Unknown, implementation not yet complete. *)
6. MAINTENANCE. (* Unknown, implementation not yet complete. *)
7. STANDARD. (* No information provided. *)
8. MEASUREMENTS. (* Unknown, implementation not yet complete. *)
9. RELIABILITY. (* Unknown, implementation not yet complete. *)
10. DEVELOPMENT METHOD. A P-code interpreter written in HP-Algol.
11. LIBRARY SUPPORT. (* No information provided. *)
Hewlett Packard HP-3000 -- Miscellaneous

See also HERE AND THERE News section under Kurt Cockrun and R. A. Lovestedt (who works at Boeing (CAD) in interactive graphics).

Also, on 7/7/75, Edward O. Thorland, Computer Center, Luther College, Decorah, IA 52101 (319/387-1043), phoned that he was ordering the P4 compiler to start an HP-3000 implementation.

Hewlett Packard HP-3000 (Santa Clara)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Ronald Danielson; University of Santa Clara; Santa Clara, CA 95093; 408/984-4482.  
2. MACHINE. Hewlett-Packard HP-3000/Series II.  
3. SYSTEM CONFIGURATION. Runs under MPE with 256K words memory.  
4. DISTRIBUTION. (# Unknown, project not yet complete. *) A very rough completion date is 78/01.  
5. DOCUMENTATION. (# Unknown, project not yet complete. *)  
6. MAINTENANCE. (# Unknown, project not yet complete. *)  
7. STANDARD. (# No information provided. *)  
8. MEASUREMENTS. (# Unknown, project not yet complete. *)  
9. RELIABILITY. (# Unknown, project not yet complete. *)  
10. DEVELOPMENT METHOD. Via Pascal-P.  
11. LIBRARY SUPPORT. (# No information provided. *)

HITACHI Hitac 8800/8700 (Tokyo)

(* See also implementation notes for IBM 360/370. *)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Teruo Hikita; Kiyoshi Ishihata; Department of Information Science; University of Tokyo; Tokyo, 113, Japan; 03-812-2111 x2947.  
2. MACHINE. Hitac 8800/8700.  
3. SYSTEM CONFIGURATION. OS7 (Hitachi). (# Minimum hardware requirements not reported. *)  
4. DISTRIBUTION. Reluctantly.  
5. DOCUMENTATION. 'Pascal 8000 Reference Manual', and 'Bootstrapping Pascal using a Trunk' are available from above address. (# Apparently no machine retrievable documentation. *)  
6. MAINTENANCE. No formal support can be promised. Bug reports are welcome.  
7. STANDARD. differences: standard procedures pack and unpack not implemented; files must be declared at main program level; extra loop control structures; "value" initialization part.

8. MEASUREMENTS. Compiler object size is about 100 kilobytes. compilation speed--about 350 lines/second. execution speed--comparable to FORTRAN-compiled objects. execution space--(# No information provided. *)  
9. RELIABILITY. Good. (# Number of sites using system and date first released not reported. *)  
10. DEVELOPMENT METHOD. A 5200 line Pascal program created by modifying Naegeli's Trunk compiler and bootstrapping it by Pascal-P. Required about 3 person-months to complete.  
11. LIBRARY SUPPORT. None -- the compiler produces absolute code, not relocatable modules.

Honeywell H316

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Robert A. Stryk; 5441 Halifax Lane; Edina, MN 55435; 612/887-4356.  
2. MACHINE. Honeywell H-316.  
3. SYSTEM CONFIGURATION. (# No information provided. *)  
4. DISTRIBUTION. (# No information provided. *)  
5. DOCUMENTATION. (# No information provided. *)  
6. MAINTENANCE. (# No information provided. *)  
7. STANDARD. A modified implementation of Concurrent Pascal, which varies from Standard Pascal.  
8. MEASUREMENTS. (# No information provided. *)  
9. RELIABILITY. (# No information provided. *)  
10. DEVELOPMENT METHOD. (# No information provided. *)  
11. LIBRARY SUPPORT. (# No information provided. *)

Honeywell 6000

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Implementor: W. Horven Gentleman; Mathematics Faculty Computing Facility; University of Waterloo; Waterloo, Ont. N2L 3G1; CANADA; 519/885-1211. Distributor: Honeywell Information Systems; 7400 Metro Blvd.; Edina, MN 55435; (# See local HIS sales office. *)  
2. MACHINE. Honeywell 6000, level 66 series. Operates under GCUS (TSS). Currently (# 7/6/75/886) a DLL TASK version is under consideration.  
3. SYSTEM CONFIGURATION. Honeywell level 66 or 6000 series with EIS. Minimum of 26k words.  
4. DISTRIBUTION. (# No information provided. *)  
6. MAINTENANCE. Supported by HIS.

7. STANDARD. Restrictions:
   - Program statement not accepted, replaced by required procedure 'main'.
   - No files with components of type file.
   - Only files of type char may be read or written.
   - Sets limited to 72 members (no sets of char).

  Extensions:
   - Files may be opened dynamically.
   - Extended file handling is available.
   - External separately compiled Pascal and FORTRAN procedures may be used.
   - Various procedures and functions to provide access to operating system.
   - 'Optional left-to-right evaluation for boolean expressions and IF statements'.
   - 'Else' clause in case statement.

8. MEASUREMENTS:
   - Compilation space--minimum of 26k words. Typical programs require less than 30k words.
   - Compilation speed--(* No information provided. *)
   - Execution space--can be as small as 4-5k words depending on the program and the
     Pascal support routines required.
   - Execution speed--(* No information provided. *)
     (* How this compares to FORTRAN and other languages not reported. *)

9. RELIABILITY. (* No information provided on reliability or number of sites using system. * ) Distributed since 76/05.

10. DEVELOPMENT METHOD. (* No information provided. *)

11. LIBRARY SUPPORT. Separately compiled Pascal and FORTRAN routines may be saved and
called from user specified libraries at run time. A post-mortem debugger is planned, but
presently (* 76/10/25 *) far from being implemented.

IBM Series 1

Gus Bjorklund, 2250 Coppersmith Square, Reston, VA 22091, reported in late June that
he had an IBM Series 1 implementation nearly complete and should be finished by 77/9.

IBM 360, 370 -- Introduction

As with DEC PDP-11s, requests for and news about IBM 360/370 implementations abound.
Last year we tracked over ten different implementation efforts. We have news for this
issue of PUG regarding improvements to the Hitac-8000 compatible compiler which has been
converted to IBM systems by the Australian AEC, as well as about the Manitoba and SUNY
Stony Brook compilers. Following these, summaries are given for other known implementations
based on news from last year.

Teruo Hikita’s University of Tokyo Hitac-8000 compiler attracted our interest last
fall when it was announced as being (1) written in Pascal, (2) very fast (as fast as the
Fortran compiler), and (3) adaptable to IBM systems. Apparently the project ran short of
resources and not much news developed until Joseph Mezzaroba (PUG #8) coaxed a copy and
with a team of graduate students it had it running in three weeks under OS/3. This summer news
came from the Australian Atomic Energy Commission (AEC) that they have finished the job
with respect to making Hitac-8000 Pascal available on IBM systems to non-commercial sites
only. So now we list Hikita’s compiler under Hitachi Hitac-8000 and replace its IBM entry
with the AEC. Joseph Mezzaroba indicates that they (at Villanova) have switched from
their version of the Hitac compiler to the AEC version.

Our thanks to W. Bruce Foulkes for sending us new and complete information on his
implementation which now, we are pleased to find, is improved, upgraded, and more
standard!

Also thanks to Richard Kieburz for sending new information plus an explanation as to
the cost of SUNY Stony Brook Pascal. It is a credit to their dedication to Pascal that they
continue to support an IBM compiler when they no longer have IBM equipment!

One final note: Thanks to Philip Malcolm (Computer Associates, Park House, Park
Street, Maidenhead, Berkshire SL6 1SL United Kingdom) who phoned twice this summer to give
us information about plans to evaluate many IBM implementations for the purpose
of writing production software (in 6 or 8 operating systems!). He found that:

1) The Technical University of Berlin has dropped their effort at a P4
implementation and has obtained Imperial College, London’s version of a P4
implementation, which “runs nicely”. (* Our only problem here at PUG(USA) is
that no one at Imperial College has told us in writing what they are doing. Here
are the names of the PUG members at Imperial College: P.W.R.Clarke,
R.A.Francis, Jeff Kramer, Stuart McMae, Greg Pugh, David Slater, Iain
Stinson, and Dave Thomas. Their address: Department of Computing and Control,
New Huxley Building, Imperial College London, 180 Queensgat, London,
England SW7 2AZ United Kingdom (phone: 01-599-5111)."

Well, how about it? *)

2) He was procuring the Australian AEC version of Hikita’s Hitac-8000 compiler.

3) He rejected the SUNY Stony Brook version.

4) He could not get the source for the Manitoba version or the source of the run
system of the Grenoble version.

5) Ruled out virtual memory, and thus the Vancouver version.

6) Still awaited news from Oslo and from Stanford.

Philip promised a followup report and evaluation and we certainly look forward to it.

--Andy Nickle

AUSTRALIAN ATOMIC ENERGY COMMISSION
NUCLEAR SCIENCE AND TECHNOLOGY BRANCH

RESEARCH ESTABLISHMENT, NEW ILLAWARRA ROAD, LUCAS HEIGHTS

TELEGRAMS: ATOBERE SYDNEY
TELEX: 24562
TELEPHONE: 512-5111

ADDRESS: ALL MAIL TO:
AAEC RESEARCH ESTABLISHMENT
PRIVATE BAG 2321, SUNSHINE, MELBOURNE, V.I.E.

IN REPLY PLEASE QUOTE: AAEC

Mr. Andy Nickle,
Editor, Pascal Newsletter,
University Computer Center,
227 Experimental Engineering Bldg.,
University of Minnesota,
MINNEAPOLIS MN 55455.

Dear Mr. Nickle,

On the 18th March, 1977, we received a copy of Pascal 8000 from
Professor Teruo Hikita, University of Tokyo, Japan. Pascal 8000 was
developed for use on a Hitachi "H" series computer, reputed to operate
under an IBM370 compatible operating system. With a few modifications
to the run-time system, we brought up Pascal on our IBM360/65 in only
a few days.

...
Basically, the compiler is excellent. The language implemented is very nearly standard Pascal with some very significant extensions. The compiler itself is written in PASCAL 8000, and produces very efficient and, in general, compact machine code. In the majority of cases, execution speed of compiled code is faster than that of a similar program compiled under FORTRAN. The original version of the compiler compiled itself in about 290K bytes.

Since March, we have been developing further the compiler for OS and VS on IBM360 and IBM370 computers. We have completely re-written the run time system in assembler (it now occupies 6K bytes instead of 36K) and in so doing have extended and implemented various features such as local file processing (in the true sense, not just temporary datasets), extended addressing (procedures can now be up to 24K bytes in length, rather than 4K), and various traceback and post-mortem dump routines. Further, files of RECFM = P1B(A) and V1B(S)(A) are supported for both input and output.

Several areas of the compiler have been restructured and extended. Procedures PACK and UNPACK are implemented, so now a true superset of standard Pascal is accepted by the compiler. Internal mechanisms of code generation were changed and new functions and standard type names added. Exponentiation has been included, parts of the lexical analyser have been rewritten, and code has been optimised in several areas. It is now possible to compile small programs in 128K, and the compiler compiles itself in 210K (corresponding figures were 176K and 286K in the original version).

We are now making this modified IBM360/370 version of PASCAL 8000 available. Distribution arrangements have not quite been finalised, however, it is envisaged that support for the system will be provided. All enquiries are very welcome.

Yours sincerely,

Jeffrey Tobias
Systems Design Section
Gordon W. Cox


In reply please quote: JMT.1mb

---

The system currently being distributed produces object code in a form suitable for its own internal loader; the code produced by the compiler can be saved for later execution (one example of this being the compiler itself), but cannot be linked with other modules. We are, however, in the final testing stages of a version that produces standard IBM linkage-editor compatible object decks, and linking to externally compiled Pascal, Fortran and Assembler routines is supported. This new version will be distributed as well as the original, as each has its own advantages.

We are enclosing a brief description of our Pascal System. All enquiries are very welcome, and order forms are available from.

Yours sincerely,

J. M. Tobin
Systems Design Section

---

PASCAL 8000 - IBM360/370 VERSION

1. Fully implements 'standard Pascal', with some very significant extensions.
2. Compiler is itself written in Pascal.
3. Total system size is relatively small. Moderately sized programs may be compiled in 128K, with the compiler able to compile itself in 210K.
4. Datasets of RECFM = P1B(A), V1B(S)(A) or V1A(A) are supported.
5. Files may be external or local. Thus, structures such as 'array of files' are available. External files are named in the program statement, local files are not. Both external and local files may be declared in a procedure at any level.
6. Arithmetic is performed in double precision.
7. Control of input and output formatting is as described in Jensen and Wirth (1975). The form is

\[
\text{<expression>}[\text{<expression>}, \ldots]\text{<expression>}
\]

where <expression> is of type set.

Elements of type packed array of char may now be read on input. Procedures read and write have also been extended to apply to both non-text and text files.

8. Procedures have a maximum size, depending on their static nesting level. This size ranges from 4K to 24K of compiled code.

9. Some of the language extensions include:
   (i) Constant definitions for structured types. It is therefore possible to have arrays, records and sets as constants.
   (ii) A 'value' statement of variable initialisation.
   (iii) A 'forall' statement of the form:

   \[
   \text{forall <control variable> in <expression> do <statement>}
   \]

   where <expression> is of type set.
(iv) A 'loop' statement, specifying that a group of statements should be repeatedly executed until an 'event' is encountered. Control may then be transferred to a statement labelled by that event.

(v) The types of parameters of procedures or functions passed as parameters must be specified explicitly, and this enables the compiler to guarantee integrity.

(vi) The 'type identifier', restriction in a procedure skeleton has been relaxed to allow 'type'.

(vii) Functions 'pack' and 'unpack' are supported, as are packed structures in general.

(viii) Exponentiation is fully supported, and is used via the double character symbol '**'.

(ix) A 'type-change' function has been introduced that extends the role of 'chr' and 'ord'.

(x) Case-tag lists may now range over a number of constants, without explicitly having to list each constant.

The range is denoted by:
<constant> .. <constant>

Thus,
4,6..10,15,30..45

is now a valid case tag list.

A default exit is also supplied via

else: <statement>

i.e. else: is a valid case tag in every case statement. This path will be used if none of the other tags match.

10. Execution errors terminate in a post-mortem dump, providing a complete execution history that includes procedure invocations, variable values, type of error, etc.

11. Object code produced by the compiler is compact and efficient. In general, execution speed of PASCAL 8000 programs is faster than that of similar programs written in FORTRAN G level.

12. Maximum set size is 64 elements.

13. Procedure 'new' is fully supported, obtaining the minimum heap requirements as specified by variant tags. Procedures 'mark' and 'release' are also supported. Procedure 'dispose' is not supported.

Reference
Restrictions
- Only the standard input and output files SYSIN and SYSPRINT are supported. All I/O is accomplished through the use of READ, READING, WRITE, WRITING, EOM, and EOF. (This is a temporary restriction. Work on file support is in progress.)
- Procedures PACK and UNPACK are not implemented.
- System to global labels are not allowed.
- SETs of characters are not supported (temporary restriction).
- Built-in procedures and functions are not accepted as actual parameters.
- The maximum static nesting of procedure and function declarations is 5.
- Program segments are restricted to 4K bytes of code.

Extensions
- Three additional scalar types are supported: SHORTINTEGER(SHRTINT), LONGREAL(LREAL), and STRING(n) lines\leq256.
- A substring operation is provided.
- Formatted input is provided and input of BOOLEAN and STRING values is permitted.
- Hexadecimal constants are supported.

8. Compiler Development
The compiler is one pass and uses a top-down parsing strategy. All semantic routines are written in PL360 (about 13,000 lines) and system interface routines in Assembler (500 lines).
The run-time routines are written in PL360 (1600 lines) with an Assembler Interface (500 lines).
Compilation speed averages 500-1000 lines of source per second on an IBM 370/168.
Considerable effort has been spent on localized optimizations in areas such as array subscripting, record field accessing and boolean expression evaluation with the aim of producing a compiler suitable for the compilation of application programs. (A 3100-line Assembler/Loader/Interpreter system has been written locally in PASCAL and is in production use on our student terminal.)
The compiler has been running on our express student terminal since January 1976.
I haven't run any speed tests recently, but execution speed seems competitive with the IBM Fortran G compiler.

9. Reliability
Good and getting better. (All problems which have been brought to my attention have been remedied.)

10. Method of Development
The compiler was hand coded. (Some routines were borrowed from the translator-writing system SYNTICS.)
The project was begun in the summer of 1972 and is still continuing. I have spent a total of 60 man-months on the project but was also teaching for 40 of those months, and have been distributing the compiler (copying and mailing tapes, etc.) for the last 20 months.
This is my first production compiler, but I now have five years experience.

11. Subprograms
The compiler produces OS-compatible object modules and uses standard IBM linkage and parameter lists in calls of external routines (Fortran, etc.). Separate compilation is not yet supported.

If people are interested in the compiler, they can write to me at the above address and I will send them a copy of my user manual, a description of the distribution tape contents, and a Software Release Agreement and order form.

Best regards,
Bruce Poulkes

IBM 360,370 (Stony Brook)

Pascal Compiler Project
Dept. of Computer Science
State Univ. of New York
at Stony Brook
Stony Brook, N. Y. 11794
July 15, 1977

Andry Mickel
University Computer Center
227 Erg Exr
University of Minnesota
Minneapolis, Minnesota 55455

Dear Andy:

Enclosed is an announcement of the newest release of the Stony Brook Pascal/360 compilers, for publication in the Pascal Newsletter. We have distributed over 100 copies of Release 1, and under our distribution policy, those who ordered Release 1 will receive Release 2 automatically.

As some of your readers may know, Stony Brook has not had an IBM 370 for over a year and a half, and we no longer have even the Univac Spectra 70 on which the Pascal/360 compiler could be executed. Our present mode of operation involves doing all machine-independent development work on a Univac 1130 at Stony Brook, then installing new developments and testing the operating system interface on an IBM 360/65 at Polytechnic Institute of New York. This arrangement is slightly inconvenient, but we work. Needless to say, we must pay for machine time and we are just breaking even (so far) on our $175 distribution fee.

Sincerely,
Richard B. Kieburz

Richard B. Kieburz

StonyBrook

STONY BROOK PASCAL/360; RELEASE 2.0 AND RELEASE 2.5

The second release of the Stony Brook Pascal compiler for IBM 360 and 370 computers is now ready for distribution.

Release 2.0 is a production compiler with facilities for linkage to externally compiled Pascal program modules and Fortran subroutines. The compiler generates IBM object modules which can be processed either by the OS/360 linkage editor or by the linking loader. Some language extensions have been installed in this release, while others are currently being implemented.

Release 2.5 is a fast, compile-look-and-go version that implements Standard Pascal without extensions. It can be installed under HASP Autobatch for economical batch processing of small jobs. The only significant restrictions imposed by 2.5 are on the maximum program size that can be compiled (dependent on the partition size allocated to the assembler). When the compiler is run under Autobatch, nonstandard files will be restricted to internal files only.

This is a restriction imposed by Autobatch, not by the
compiler. Release 2.0 incorporates all improvements that were made during the lifetime of Release 1. In addition, the irreducible overhead to compile a trivial program has been reduced, by simulating the sequential access file I/O used for inter-pass communication with main-storage files. The current minimum storage requirement to run release 2.8 is slightly less than 130 Kbytes. Further reduction in the minimum partition size is a goal of future updates.

Release 2.0 implements Standard Pascal, including external module and standard modules defined using the GOTO access method, with the following enhancements:

1. **Module definition and linkage** A program module consisting of global type and data, functions and procedures, may be given an attribute external in the program heading. This informs the compiler that it is to be regarded as an ancillary compilation module, and the main program body is to be ignored. A procedure or function that is declared external is callable from anywhere in the program, including the main program body. External references are declared by giving a procedure or function declaration in which the body is replaced by the external name. A main program can also contain entry procedures and external references. Global data that are to be shared among modules are declared in a section called extvar. Otherwise, global data are private to the program module in which they are declared.

Each compilation prepares an output file called a Pascal Object Module, which contains not only code and data, but also a symbol table describing the names, types, and storage given to its entry points and external references. A new system called Lexical Order, based on the lexical order of the symbols in the program's source module, links external references, and checks them for type-correctness, outputting an IBM module object. The IBM linker, which is part of the object module, performs final relocation and linkage to the standard environment, including any callable Fortran library routines that may have been specified.

We believe this new facility will prove to be a powerful tool for modular design and testing of large programs. The post-mortem-dump diagnostic facility is preserved throughout the linkage process, if it has been requested as a compiler option.

2. **Parametric constants** (to be implemented in Release 2.1)

In a const declaration, a constant may be given a declared type, by a syntax extension. A constant whose type is not one of the predefined scalar types is declared on the line with its value part declared as external, meaning that the actual value is unknown at compile time. The value must be supplied at linkage time, either from a constant of the same name (and type) declared in a main program module, or as a named parameter to the program linker.

The principal reason for this extension is that it allows the declaration of subrange types whose bounds can be altered without recompilation; thus array and set types can have their sizes adjusted to the demands of an application merely by re-linking a program. This scheme is admirably a compromise between the desire to allow Algol 60 dynamic array sizes, the need to perturb the type formation rules of Pascal. Only the experience of users will tell whether or not it is adequate. For more details on the scheme see [1].

3. **Storage management** (to be implemented in Release 2.1)

Effective management of heap storage has been a problem in Pascal, exacerbated by the type violation allowed with subrange types. Conventional semantically-garbage collection is frustrated by the lack of certainty as to whether or not a reference to a storage descriptor still holds. We have implemented a storage management scheme using two-level indirection (see [2] for a general discussion) in which a pointer actually points to an index into a storage descriptor table, invisible to the user. Descriptors contain a heap address, a hold count, and a pointer to a storage template. This has the advantage that the heap itself contains no absolute addresses, and therefore, most storage reclamation can be done simply by compaction. References to dangling pointers are immediately detectable, and it is therefore practical in implementation to use the standard procedure Dispose, allowing a programmer to control the release of dynamically allocated variables.

Obviously, it is impossible for inaccessible, but cyclic data structures to exist undetected and to congest the descriptor table. When this condition occurs, the inaccessible descriptor storage can be reclaimed by semantically-garbage collection, within the array of fixed-length descriptors.

4. **String processing** (to be implemented in Release 2.1)

A new, built-in type, String, has been added. Values of this type are used for representation of source listing, operator names, identifiers, and data references. Operations on strings are realized using the go-to lexical order. Operators are all realized by means of predeclared functions; no new operator symbols have been added. These functions include:

- function Concatenate (A : String; B : String) : String;
- function Length (A : String) : Integer;
- function Substr (A : String; B : Integer; C : Integer) : String;

String values may be compared for equality or inequality (using lexical order) by applying the standard Pascal comparison operators. The symbol denotes the constant, empty string, Procedures Write and Readln. Strings to be input must be quoted by apostrophes.

Corrections are allowed for assignment from any expression of types array or Char to a variable of type String. Corrections in the reverse direction are not allowed.

String processing is provided by a Pascal-linkable library module, which need not be included in the runtime support package if it is not needed. Two implementation methods are compared. The machine-dependent and very maximum variant is selected.

5. **Procedural type Alfa** has been defined to be:

Alfa : packed array [1..30] of Char

This type is principally intended to permit efficient reading of fixed-length, 8-byte records. If the condition Eoln on the variable is not true, then the variable is read. This type is also useful for reading the 8th byte, a read error occurs.

6. **Constructors for typed constants**

A new system of overloaded operators has been defined, as illustrated in the following example:

```
const
  A : Alfa := "ABC"
  B : Char := 123

var
  C : Char

begin
  C := A[1];  // C := 'A'
end;
```

Constructors for typed constants may now be declared. The constructor syntax uses parenthesis to reflect the internal structure of the declared type. In order to allow localization of possible errors in defining a constructor, the constructor syntax is limited to one dimension only. The constructor may be created for any type, except file or pointer. For structured types, each literal value given as a component is checked for compatibility with the declared type of the corresponding field. Parametric constants are not allowed as components of a constructor, nor can the type of a constant depend upon any parametric constant.

7. **Default clause on case statements**

We have allowed the end of a case statement to be optionally replaced by the clause otherwise (statement). Also, ranges of constants are allowed as case labels. The cardinality of a label range is not limited by any implementation restriction, but only by the cardinality of its base type. The implementation uses a combination of branch jumps and jump tables; a more detailed discussion of this technique can be found in [1].

8. **Diagnostic aids**

The two principal run-time diagnostic aids provided by the Pascal/360 compiler are a symbolic, post-mortem dump of active variables, and an execution-count profile printed on a formatted source listing. The profiles have proven to be valuable tools for analysis and debugging of Pascal programs, and for compiler debugging as well. The post-mortem dump has been extended to include variables in compiled modules, although the execution profile will only apply to the main program module.

The efficiency of run-time subrange checks has been improved slightly from that of Release 1, and a standard procedure Tolt has been added, to allow a post-mortem dump to be included under program control.

9. **Cross-reference listings**

A new compiler option provides a cross-reference listing of all identifiers and the source program line numbers where they occur. The listing also gives the name of the block in which each identifier is declared, this option is useful in maintenance and documentation of large programs.

Future development efforts will be directed towards:

a) eliminating any bugs that may have been added along with the new compiler features;

b) an optional code optimizer;

c) reducing the main storage requirement of the compilers;

d) further enhancement of the diagnostics.

References:


IBM 360, 370 (Grenoble)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. J. P. Fauche, Departement Informatique, IREP, Boite Postale 47, F-38040 Grenoble Cedex, France.
2. MACHINE. IBM 360/67, 370/148.
3. SYSTEM CONFIGURATION. Runs under OS/MVT (360/67) and VS/MFT (370/148). Requires 220K for self-compilation.
4. DISTRIBUTION. Distribution is via 9 track, 800 bpi magnetic tape.
5. DOCUMENTATION. The implementation is described in a supplement to the User Manual.
6. MAINTENANCE. (* no information *)
7. STANDARD. Deviations are described in the documentation.
8. MEASUREMENTS. The standard compiler (6000 lines of Pascal) compiles in 105 CPU seconds; an enhanced compiler compiles in 84 seconds.
9. RELIABILITY. (* no information *)
10. DEVELOPMENT METHOD. (* no information *)
11. LIBRARY SUPPORT. Assembler procedures are supported.

IBM 360, 370 (Stanford)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Sassan Hazeghi, Computation Research Group, SLAC, P.O.Box 4349, Stanford, CA 94305.
2. MACHINE. IBM 360, 370.
3. SYSTEM CONFIGURATION. (* no information *)
4. DISTRIBUTION. The entire system is available to the public (as is).
5. DOCUMENTATION. (* no information *)
6. MAINTENANCE. No maintenance is promised.
7. STANDARD. Implements the Pascal-P2 (May, 1974) subset, with a few minor extensions.
8. MEASUREMENTS. The system self-compiles in 130K bytes, 24K of which is returned to the operating system for I/O buffers.
9. RELIABILITY. (* no information *)
10. DEVELOPMENT METHOD. Developed from Pascal-P2. P-code was first translated to assembly code by macros; a P-code translator was then written in Pascal. The P-translator can produce either assembler code or a standard OS object module.
11. LIBRARY SUPPORT. (* no information *)

IBM 360, 370 (Socorro, New Mexico)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Implementors: Jan V. Garwick, Paul Verillat, and Robert Knight, Computer Center, New Mexico Tech, Socorro, New Mexico 87801. Distributor: Tom Narker, Computer Science Department, New Mexico Tech, Socorro, New Mexico 87801 (505/835-5126). Direct non-distribution questions to Robert Knight.
2. MACHINE. IBM 360, 370 series.
3. SYSTEM CONFIGURATION. OS operating system.
5. DOCUMENTATION. (* no information *)
6. MAINTENANCE. (* no information *)
7. STANDARD. The following are not supported: gotos and labels; unpacked arrays; and sets of characters.
8. MEASUREMENTS. (* no information *)
9. RELIABILITY. Results of one month of testing were good (76/9/20).
10. DEVELOPMENT METHOD. Designed by Jan Garwick and implemented in PL360 using GPM.
11. LIBRARY SUPPORT. (* no information *)

IBM 360, 370 (Oslo)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Ivar Laberg, Computer Department, University Hospital Oslo, Rikshospitalet, Oslo 1, Norway (471 20 10 50).
2. MACHINE. IBM 370/125.
3. SYSTEM CONFIGURATION. DOS/VS operating system.
4. DISTRIBUTION. (* no information *)
5. DOCUMENTATION. (* no information *)
6. MAINTENANCE. (* no information *)
7. STANDARD. A number of extensions are being considered, including: interface to all secondary storage access-methods; external procedures written in other languages; and "external records" (functionally equivalent to "named common" in Fortran).
8. MEASUREMENTS. (* no information *)
9. RELIABILITY. (* no information *)
10. DEVELOPMENT METHOD. Based on Pascal-P.

11. LIBRARY SUPPORT. (* no information *)

IBM 360,370 (Vancouver)---------------------

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Barry W. Pollack and Robert A. Fraley, Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada V6T 1W5 (604/228-6794 or 604/228-3061).

2. MACHINE. IBM 370/168.

3. SYSTEM CONFIGURATION. The current version runs under the MTS operating system. The monitor may be modified with minimal effort to run under VS, OS, etc. Standard OS object modules are generated. The translator requires about 320K bytes of storage. Division of the compiler into overlays for non-VH systems would be possible.

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


6. MAINTENANCE. No policy has been decided. 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. 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.
The is a case default label: "O".
Optional "T" allowed before else.
"(...)" may be used instead of "[...]".
The character ~ has been retained.
Packed is ignored.
Input of character strings using read is allowed.
Support of EBCDIC characters "k", "", and (logical not sign). (* Sorry, we use ASCII at PUG News. *) Use "..." for comments.
Value section exists for variable initialization.
Hexadecimal integers are supported.
FORTRAN subroutines may be called. A return code is available in the pre-declared variable code.
Direct access files are supported.
Additional built-in functions include: min, max, substr (using constant length), position (direct access files), I/O interface functions and extensions to reset and rewrite, and insert for data packing.

Restrictions:

Sets are limited to 32 elements (0..31).
Program heading is not used.
Files may not be components of other structures.
Set constructors may not include <expression>, <expression>
Input is initially ~ instead of the first character of the file. This is transparent when read is used.

Projected extensions:

McCarthy II,
y and and lower precedence than relations.
"Usual" precedence used throughout.
Sets over the range 0..255.
Better control of input and output formats.

8. MEASUREMENTS. The compiler 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 C code. The translator 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. The compiler requires 320K bytes of core.

9. 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. DEVELOPMENT METHOD. The original translator was developed by Wirth and several graduate students at Stanford University as a partial rewrite 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 (* anonymous? *) graduate student.

11. LIBRARY SUPPORT. Fortran routines can be called. The compiler generates standard OS object modules.

IBM 1130---------------------

We have heard of two possible implementations, by:

(1) H. Sandmeyr, Neu-Technikum, CH-9470 Buchs, Switzerland (085/6 45 24).
(2) Fred Powell, Innovative Management Systems, 865 Middlebrook Av., Staunton, Virginia (703/885-6950). (Fred was formerly at Mary Baldwin College.) "Little has been done so far," according to Fred (76/12/10).

ICL 1900 (Belfast) - HK2.

---------------------

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Jim Welsh, Colum Quinn, and Kathleen McNane, Department of Computer Science, Queen's University, Belfast BT7 1NN, Northern Ireland, U.K. (* No phone number provided. *) Enhancements by David Watts and Bill Findlay, Computer Science Department, University of Glasgow, Glasgow G12 8QQ, Scotland, U.K. (* No phone number provided. *)

2. MACHINE. ICL 1900

3. SYSTEM CONFIGURATION. Has been installed under George 3, George 4, Executive, MAXMOP, and COOP operating systems. Requires 32K.

4. DISTRIBUTION. (* no information *)

---------------------

PASCAL NEWS #9 & #10

SEPTEMBER, 1977

PAGE 101
5. DOCUMENTATION. A clearly written machine retrievable Supplement to the Revised Report, dated 7/7/23.

6. MAINTENANCE. (* no information *)

7. STANDARD. Primarily implements the Revised Report; exceptions include (a) files not allowed as components of structured types, and (b) non-discriminated variant records are not allowed. A six bit character set is used. Sets may have at most 48 elements. A value initialization part is implemented.

8. MEASUREMENTS. Compares favorably to Fortran, requiring about 32K to compile. Generated code is better than that produced by the old 1990 Pascal compiler. (* Compilation speed not reported. *)

9. RELIABILITY. Reported to be good. The compiler is in use at about 50 sites.

10. DEVELOPMENT METHOD. This compiler resulted from a complete rewrite of the old ICL 1990 compiler. The new compiler is designed for portability, with a clean separation between semantic analysis and code generation.

11. LIBRARY SUPPORT. Allows access to Fortran routines.

ICL 2970, 2980 (London)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. John Reynolds and Jules Zell, Department of Computing and Control, Imperial College, London SW7, U.K. (* No phone number provided. *)

2. MACHINE. ICL 2970, 2980 series.

3. SYSTEM CONFIGURATION. (* no information *)

4. DISTRIBUTION. Contact David Joslin, Sussex University Computer Centre, Brighton, Sussex, U.K.

5. DOCUMENTATION. (* no information *)

6. MAINTENANCE. (* no information *)

7. STANDARD. Presumably similar to the ICL 1900 M22 compiler.

8. MEASUREMENTS. Code generated is fairly compact, the compiler itself producing 80000 bytes. This is better than the 2900 standard compiler. The CDC Pascal 6000 compiler compiles the 2900 compiler on a CDC 6400 in 82 seconds. The ICL compiler self-compiles on the 6400 in 100 secs. Running on a 2900, the 2900 compiler self-compiles in 360 seconds. John Reynolds tells us, "I've determined that almost all time required for a compilation on the 2900 is just burnt up by the system and that hardly any time at all goes in the actual act of code generation." (7/7/78) (* Execution speed of generated code not reported. *)

9. RELIABILITY. The compiler has been extensively tested and seems to work fairly well. (* Date of first release and number of sites using system not reported. *)

10. DEVELOPMENT METHOD. Based on the ICL 1990 M22 compiler, with code generators rewritten. Poor performance of the ICL 2970 system led to development on a Control Data 7600 using Zurich's Pascal-6000.

11. LIBRARY SUPPORT. (* no information *)

Intel 8080 (INSITE)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Implemented by Thomas A. Wolander, 1012 Smith Ave., Campbell, CA 95008 (408/378-5785). Distributed by INSITE, Intel User's Library, Microncomputer Division, 3085 Sowers Ave., Santa Clara, CA 95051 (408/246-7501 x2948).

2. MACHINE. Intel 8080A using the Intel Intellec Microcomputer Development System.

3. SYSTEM CONFIGURATION. Operating system: Intel MDS ISIS-II. Hardware: 64K bytes of RAM and dual floppy disks.

4. DISTRIBUTION. The software is distributed on two soft-sectored diskettes, and includes the binaries and sources.


6. MAINTENANCE. Bug reports will be accepted.

7. STANDARD. Implements Brinch Hansen's Sequential Pascal, except for floating point (which is under development - 7/7/22).

8. MEASUREMENTS. The virtual machine interpreter is 1300 lines of code (PL/M-80) and 10K bytes. Compilation speed is 30 lines/minute. (* Execution speed and size of generated code not reported. *)

9. RELIABILITY. Will self-compile and has been used successfully by students. (* Number of sites using system not reported. *)

10. DEVELOPMENT METHOD. An interpreter (PAS80) was written in PL/M-80, and emulates a 16-bit machine. The implementation required about "2 non-months-of-eveings" and was accomplished in the implementor's spare time. The implementor was familiar with the process of implementing the virtual machine. "Credit for the ease of implementation is due to Per Brinch Hansen who developed the virtual machine."

11. LIBRARY SUPPORT. (* no information *)

Intel 8080 (Minneapolis)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Peter Zechmeister, University Computer Center: 227 Exp Eng, University of Minnesota, Minneapolis, MN 55455 (612/373-4181).

2. MACHINE. Intel 8080.

3. SYSTEM CONFIGURATION. An operating system is included with the implementation. The minimal hardware required is an I/O device (TTY) and about 16K bytes for the compiler.

4. DISTRIBUTION. Has not been determined.

5. DOCUMENTATION. In progress.

6. MAINTENANCE. Under development.

7. STANDARD. The implementation is called Tiny Pascal (TP). It does not provide a number of standard features due to size constraints.

8. MEASUREMENTS. The bootstrap cross-compiler runs at 2400 lines/minute on a CDC 6400. The TP compiler itself loads in about 14K.
9. RELIABILITY. The reliability of the compiler is excellent. (* Number of sites using system not reported. *)

10. DEVELOPMENT METHOD. Based on the PLO compiler by Niklaus Wirth. Modifications were made to implement "variable types, Pascal statements, code generation, and register mapping." A cross-compiler running on a Control Data 8600 has been used to develop the Tiny Pascal (8080) compiler, which was not complete as of PUGN #.

11. LIBRARY SUPPORT. None.

**Intel 8080a (San Diego)**

*See DEC LSI-11 (San Diego), above.*

**Interdata 7/16**

Two possibilities to check out:

Mike Ball (see Interdata 8/32 for address) has Concurrent and Sequential Pascal cross-compilers running on the U100 generating code for the Interdata 7/16.

Rod Steel, Tektronix, MS 60-436, PO Box 500, Beaverton, OR 97707 (503/638-3411 x2515), reported a year ago that he might attempt to bring up Pascal on the 7/16. No news since then.

**Interdata 7/32**

*See Kardios Duo 70, below.*

**Interdata 8/32 (Parkville, Australia)**

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Mike Ball, Code 632, Naval Ocean Systems Center, San Diego, CA 92152. (* No phone number reported. *)

2. MACHINE. Interdata 8/32.

3. SYSTEM CONFIGURATION. (* no information *)

4. DISTRIBUTION. (* no information *)

5. DOCUMENTATION. (* no information *)

6. MAINTENANCE. (* no information *)

7. STANDARD. (* no information *)

8. MEASUREMENTS. (* no information *)

9. RELIABILITY. (* no information *)

10. DEVELOPMENT METHOD. (* no information *)

11. LIBRARY SUPPORT. (* no information *)

**Interdata 8/32 (Kansas)**

(*) See Mike's letter in the OPEN FORUM section *)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Mike Ball, Code 632, Naval Ocean Systems Center, San Diego, CA 92152. (* No phone number reported. *)

2. MACHINE. Interdata 8/32.

3. SYSTEM CONFIGURATION. (* no information *)

4. DISTRIBUTION. "It will not be available for distribution for at least several months." (7/7/6/15)

5. DOCUMENTATION. (* no information *)

6. MAINTENANCE. (* no information *)

7. STANDARD. Brinch Hansen's Sequential and Concurrent Pascal.

8. MEASUREMENTS. (* no information *)

9. RELIABILITY. (* no information *)

10. DEVELOPMENT METHOD. (* no information *)

11. LIBRARY SUPPORT. (* no information *)

**KANSAS STATE UNIVERSITY**

August 11, 1977

Dear Mr. Mickel:

As reported by Mike Ball in the Pascal Newsletter #7, we have transported the Brinch Hansen Concurrent Pascal system from the PDP 11/45 to the Interdata 8/32. This implementation in its present form uses an interpreter for a slightly modified version of the abstract code as distributed by Brinch Hansen. I am enclosing for your information a copy of the Implementation manual for the system and the implementation checklist as requested for the Implementation notes section of the Pascal newsletter.

Sincerely,

David Neal
Research Assistant

DCN: tlb
Enclosure (1)
1. Implementors:
   David Neal, Gary Anderson, Jim Ratliff, and Virgil Valentine.
   Department of Computer Science
   Kansas State University
   Manhattan, Kansas 66506

   Distributors:
   Interchange (Interdata Users Group)
   Interdata, Inc.
   Oceanport, New Jersey 07757

2. Hardware:
   Interdata 7/32 or 8/32.

3. Operating System:
   OS/32-MT, minimum partition size 72.75 K, disk storage required,
   floating point support necessary.

4. Method of distribution:
   Nine track tape -- details available through Interchange.

5. Documentation:
   KSU Implementation of Concurrent Pascal -- A reference Manual, KSU
   Technical Report CS 76-16 will be provided with the implementation.
   The availability of these references is a necessity.

6. Maintenance Policy:
   None

7. Fully implements Concurrent Pascal and Sequential Pascal (SPASCAL)
   a subset of Standard Pascal.

8. Sequential and Concurrent Pascal programs are executed by a code
   interpreter written in Interdata CAL assembler language. This
   interpreter as well as the Concurrent Pascal Kernel are provided in
   source and object. The system consists of about 5000 source lines
   and requires a library segment of 7.50 K for execution. Pascal
   source is translated into code by the Hartman Compilers which are
   written in Sequential Pascal (SPASCAL). The source and object of
   these compilers are also contained in the package. Microcode
   routine for virtual instruction decode are included for the 8/32.

9. Reliability:
   Excellent -- all errors detected at KSU have been traced to hardware.

10. Method of development:
    Translated from the Brinch Hansen PDP 11/45 implementation. The
    system was moved with an approximate outlay of 4 person-months of
    experienced graduate student effort.

11. Sequential Pascal programs may call one another in arbitrary,
    recursive fashion using the interfaces of the SOLO operating
    system (which is written in Concurrent Pascal). No provision
    is made for FORTRAN or any other language. The utility programs
    of the SOLO operating system include the Sequential and Concurrent
    Pascal Compilers, a text editor similar to Interdata's OS-Edit, and
    the source code configurator program mentioned by Mike Ball (Pascal
    Newsletter #7 p. 29). All programs are maintained by the SOLO
    file system and appears to OS/32-MT as a single contiguous file.

Kardios Duo 70

See IBM 360, 370, above.

The Kardios Duo 70 consists of an Interdata 7/32 modified by Kardios Systems Corp.,
3820 Courtleigh Dr., Randallstown, MA 21133 (301/542-6826). The machine includes firmware
which emulates both Interdata and IBM 360, 370 systems. The system is designed to concurrently execute both Interdata and IBM software. According to Kardios, most software such as the IBM Pascal implementations will run on the Duo 70 with little or no modifications. The changes most often required are: use Interdata CSS instead of IBM JCL; change IBM file access calls to Interdata access calls (this is only necessary in the few cases where the IBM file access methods are not supported by Interdata). The Duo 70 will execute 360, 370 object modules produced by a compiler with no changes at all. Kardios reports that their customers have reported very little trouble in modifying 360, 370 software to run on this system.

Mitsubishi MELCOM 7700.

See IBM 360, 370, above.

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Masato Takeichi, formerly at Dept. of Math.
   Engineering and Instr. Physics, University of Tokyo, Bunkyo-ku, Tokyo 113, Japan. Present
   address: Department of Computer Science, University of Electro-Communications,
   1-5-1 Chofugaoka, Chofu-shi Tokyo 182, Japan.

2. MACHINE. MELCOM 7700, 256K bytes.

3. SYSTEM CONFIGURATION. BPM. (* Minimum hardware required not reported. *)

4. DISTRIBUTION. (* no information provided. *)

5. DOCUMENTATION. See "Pascal Implementation and Experience" by Masato Takeichi, Journal
   of the Faculty of Engineering, University of Tokyo 34:1, pp 129-136.
6. MAINTENANCE. (* no information provided. *)

7. STANDARD. Comparable to CII IRIS 80 implementation by Mancel and Thibault.

8. MEASUREMENTS. Self compiles in 150 sec. and 150 Kbytes (108K for code, 10K for monitor, 32K for data). Execution times, relative to Fortran, are given in the following table.

<table>
<thead>
<tr>
<th>Extended Fortran IV</th>
<th>Pascal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix multiply</td>
<td>1.90</td>
</tr>
<tr>
<td>Sort</td>
<td>1.75</td>
</tr>
<tr>
<td>Additive partition</td>
<td>0.48</td>
</tr>
<tr>
<td>Character count</td>
<td>0.34</td>
</tr>
</tbody>
</table>

9. RELIABILITY. Was first released in April, 1976, with the author using for several months before that. Several compiler errors have been corrected. (* Number of sites not reported *)

10. DEVELOPMENT METHOD. The compiler is based on the CII IRIS 80 compiler by Hance and Thibault, with modified code generation. The monitor and library procedures were rewritten to interface with BPM.

11. LIBRARY SUPPORT. (* no information *)

---

MITS Altair 680b

(* See implementation notes for Motorola 6800. *)

MOS Technology 6502 (San Diego)

See DEC LSI-11 (San Diego), above.

Motorola 6800 (San Diego)

See DEC LSI-11 (San Diego), above.

Motorola 6800 (St. Paul)

---


2. MACHINE. Designed for the MITS Altair 680b, based on a Motorola 6800.

3. SYSTEM CONFIGURATION. Requires 32K bytes and a TTY. No disk needed.

4. DISTRIBUTION. (* no information *)

5. DOCUMENTATION. (* no information *)

6. MAINTENANCE. (* no information *)

7. STANDARD. The following are not supported: files (except TTY input and output), and get, put, reset, rewrite; with and goto; sin, cos, arctan, exp, ln, sqrt, pack, and unpack.

8. MEASUREMENTS. Compiler code occupies 24K bytes, the interpreter requires 3K bytes.

9. RELIABILITY. Seems to be excellent. Not yet released.

10. DEVELOPMENT METHOD. Based on Pascal-P2, cross-compiled first from a Univac 1100 (using San Diego Pascal), and later from a CDC 6400. As of 7/11/84, about 2 man-months had been invested. The compiler source is about 2200 lines. The cross-compiler has been designed to be independent of the host-machine's character set. The interpreter could be implemented on other 8-bit machines with minimal work.

11. LIBRARY SUPPORT. (* no information *)

---

Nanodata QM-1

(* no information *)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Dennis Heimbigner, TRW DSSG, Mail Station R3/1072, 1 Space Park, Redondo Beach, CA 90278 (213/535-0833).

2. MACHINE. Nanodata QM-1.

3. SYSTEM CONFIGURATION. 256K words nanostore; 8K words control store; 9755 (50M byte) disk; TASK version 1.04.02 (or later); PROD version 2.04.01 (or later). Optional: Card Reader, Printer (very desirable).

4. DISTRIBUTION. "Release by TRW is currently under consideration. Inquiries are welcome." (7/7/3/17)

5. DOCUMENTATION. Brinch Hansen's SOLO manuals (not available from TRW); machine readable document describing the implementation and ways to modify it.

6. MAINTENANCE. (* no information *)

7. STANDARD. (* no information *)

8. MEASUREMENTS. Executes at about one-third the speed of the PDP11/45 (SOLO) system. (* Space requirements not reported. *)

9. RELIABILITY. (* no information *)

10. DEVELOPMENT METHOD. The Concurrent Pascal system kernel was programmed in micro-code in 6 months of part-time work. Half of that time was spent on I/O drivers.

11. LIBRARY SUPPORT. (* no information *)

---

NCR Century 200

(* no information *)

Jack Laffe, 320 19th Ave. S., Minneapolis, MN 55454 (612/336-4946) tells us (7/7/08/30) that he is writing a Pascal compiler in Neat 3 for the Century 200.
Norsk Data NORD-10 (CERN)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. David L. Bates and Robert Caillian, PS/CCI Group, CERN, 1211 Geneva 23, Switzerland. (Tel. 41-98-11)
2. MACHINE. Norsk Data NORD-10.
3. SYSTEM CONFIGURATION. SINTXAN III operating system.
4. DISTRIBUTION. "Anyone is welcome to receive a copy of our system." (77/1/19)
5. DOCUMENTATION. (* no information *)
6. MAINTENANCE. (* no information *)
7. STANDARD. (* no information *)
8. MEASUREMENTS. It takes 15 minutes to compile the compiler. (* Space requirements not reported. *)
9. RELIABILITY. (* no information *)
10. DEVELOPMENT METHOD. From Pascal-P4. P-code is assembled and then interpreted by an assembly language program.
11. LIBRARY SUPPORT. (* no information *)

Norsk Data NORD-10 (Oslo)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Andora Fjeldsgaard, Petter Gjerull, Stein Gjessing, Jan Husemoen, Kefil Moen, and Terje Noodt, Computing Center, University of Oslo, Blindern, Oslo 3, Norway. (* No phone number provided. *)
2. MACHINE. Norsk Data NORD-10, using 2 64K memory banks.
3. SYSTEM CONFIGURATION. MVS operating system.
4. DISTRIBUTION. (* no information *)
5. DOCUMENTATION. The implementation is described in "Rapport om implementering av Pascal pa NORD-10", University of Oslo, April 1976. A machine readable document describes changes and improvements to the implementation as they are made.
6. MAINTENANCE. It is expected that the system will be improved and changed frequently in the near future. Error reports are invited, and may be given to any member of the PASCAL group.
7. STANDARD. Files (except input, output, PRS, and PRF) and formal procedures are not implemented. Sets may have 64 elements; parameters and local variables (except arrays and records) may occupy at most 233 16-bit words in any procedure; strings may be at most 16 characters long.
8. MEASUREMENTS. (* no information *)
9. RELIABILITY. (* no information *)
10. DEVELOPMENT METHOD. Developed from Pascal-P4.
11. LIBRARY SUPPORT. (* no information *)

Prime P-400

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Phillip H Enslow, School of Information and Computer Science, Georgia Tech., Atlanta, GA 30332 (404/894-3187).
2. MACHINE. Prime P-400.
3. SYSTEM CONFIGURATION. Virtual memory operating system.
4. DISTRIBUTION. (* no information *)
5. DOCUMENTATION. (* no information *)
6. MAINTENANCE. (* no information *)
7. STANDARD. (* no information *)
8. MEASUREMENTS. (* no information *)
9. RELIABILITY. (* no information *)
10. DEVELOPMENT METHOD. Bootstrapped from Pascal-P4 during 1976-77.
11. LIBRARY SUPPORT. (* no information *)

SEMS T1600

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Alain Tisserant, Departement Informatique de l'INPL, Ecole des Mines, Parc de Sauropt, 54042 Nancy Cedex, France. (Tel. (28) 51 42 32)
2. MACHINE. SEMS T1600 and SOLAR 16/05/40/65.
3. SYSTEM CONFIGURATION. 8050 operating system. Hardware: MTS16; PRD or MUR disk; minimum 16K words of core memory.
4. DISTRIBUTION. Not yet available (77/2/2). Will be distributed by IMIA.
5. DOCUMENTATION. All available documentation is written in French. (* We don't know what is available. *)
6. MAINTENANCE. (* no information *)
7. STANDARD. "Fully implements standard Pascal; also compatible with IRIS 80 Pascal compiler." Extensions include: character strings; loop-exit-end statement; I/O of sets and scalars; sets of any interval of integers.
8. MEASUREMENTS. (* no information *)
9. RELIABILITY. "Expected to be excellent!"
10. DEVELOPMENT METHOD. The compiler(s) is written in Pascal. A two pass scheme uses an adaptation of P-code as an intermediate language. The P-code was adapted for non-stack, 16-bit, based addressing and accumulator machines. The first pass can be parameterized, and the second pass can be rewritten to port the compiler to other machines. An automatic segmentation mechanism allows compilation and execution of large programs (such as the compiler) with small memory requirements.
11. LIBRARY SUPPORT. The implementation allows separate compilation, as well as insertion of ASM and Fortran routines.
Dear Andy:

Enclosed you find some details about our brand new Sequential-PASCAL-Machine for the SIEMENS 330 Process Control Computer, which has been implemented by me in 2.5 manmonths. The SIEMENS-Version runs under the Real-Time OS ORG330PP2 or ORG330K and is full compatible to our VARIAN-V75-PASCAL, which we received from VARIAN Munich and which was modified and extended by me in few weeks.

The language is identical to Sequential PASCAL by P. Brinch Hansen and Al Hartmann for the Concurrent PASCAL Machine on PDP11/45. So the Concurrent PASCAL Compiler, after few modifications, is now running on our SIEMENS 330 as a Cross-Compiler for the Concurrent PASCAL Machine.

We still have not so much experience with PASCAL-applications, but while implementing and modifying these two systems, we have found no Compiler-Errors at all. The Code- and Runtime-Efficiency is good compared with RT-Fortran and will allow RT-application in PASCAL.

As prerequisite for a good flexibility, the user himself is able to extend the PASCAL-system using the external procedure interface of the system.

Although the distribution and maintenance policy of my Company is not yet fixed, I enclose some information about our PASCAL-Systems. I think that this information is interesting mainly European users, but our interest is now to receive complete PASCAL-Programs from other users, mainly a good source editor, intender and cross-reference would be welcome. Additionally we are looking for PASCAL on a DECLSIII or INTEL8080 Micro-Computer.

Sincerely yours

[Signature]

Kernforschungszentrum Karlsruhe
Institut für Datenverarbeitung in der Technik
Karlheinz Kapp

---

Sequential PASCAL

1) Implementor/Distributor
   VARIAN
   Data Machines
   D-8000 München
   W-Germany

2) Machine
   VARIAN V75
   SIEMENS 330

3) Op. System
   VORTEX II E/F
   ORG330PP2/ORG330K
   SEGSYS

4) Min. Config.
   64K Memory
   FPP/WCS
   CR/LP/DISK/MT
   TTY
   20K
   20-22.5K

5) Distribution
   ask Varian
   Manual in English
   Users Guide in German
   no Standardprocedures

6) Maintenance
   ask Varian
   on request

7) Standard PASCAL
   I/O-System is different
   7-Pass-Compiler written in PASCAL
   8-Pass-Compiler written in PASCAL

8) Type of Program
   Interpreter and I/O-System
   written in Macro/Assembler

9) Code efficiency
   3 W PASCAL-Code/source line or
   1 W PASCAL-Code/9 signif. chars
   3K
   1.8K
10) Speed

Compiler compiles typical PASCAL-source-text at a rate of 150 lines/min or 500 significant chars/sec.

For more information about Sequential PASCAL read the publications about Concurrent PASCAL.

Siemens 4004, 7000 (Munich)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Manfred Sommer, SIEMENS AG, Department D AP GE, Postbox 70 0078, D-8000 Munich, West Germany (089-722-61276).
2. MACHINE. Siemens 4004 and 7000 series. Also RCA Spectra 70 (VMOS).
3. SYSTEM CONFIGURATION. BS2000 operating system.
4. DISTRIBUTION. Contact the implementor.
5. DOCUMENTATION. A User's Manual (German) is available.
6. MAINTENANCE. (* no information *)
7. STANDARD. Appears to conform fully with the PASCAL User Manual and Report. Character set is EBCDIC. Sets may have 256 elements (allowing set of char).
8. MEASUREMENTS. "Code produced seems to be much faster than the code produced by the standard Fortran compiler." Compilation speed is 40 lines per second on a 4004/151 and 100 lines per second on a 7000/7353 (roughly equivalent in power to an IBM 370/155 or CDC 6400). In a dozen or so benchmark programs times were comparable with CDC-6400 Pascal.
9. RELIABILITY. Over 18 sites using this version.
10. DEVELOPMENT METHOD. Based on Pascal-P4.
11. LIBRARY SUPPORT. Generated code may be put into a standard module library. Additional procedures are available for interfacing to the operating system.

Siemens 4004 (Darmstadt)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. H.-J. Hoffmann, Fachbereich Informatik, Techn. Hochschule, Steubenplatz 12, D-6000 Darmstadt, Germany. (* No phone number provided. *)
2. MACHINE. Siemens 4004/157.
3. SYSTEM CONFIGURATION. (* no information *)
4. DISTRIBUTION. (* no information *)
5. DOCUMENTATION. (* no information *)
6. MAINTENANCE. (* no information *)
7. STANDARD. (* no information *)
8. MEASUREMENTS. (* no information *)
Texas Instruments TI-ASC.

Philip Bergstresser (see HEKE AND THERE News section) phoned 77/05/26 to correct our information in PUGN #8. The PDL (Production Development Language) system TI implemented included a superset of Pascal and a library management system. This included software tools, a check for matching source and binary module interfaces, procedures recompiled independently with scope, complete reversible overlay process, cross reference and instrumentation code. Documentation is available from Bill Bixler at TRW Huntsville. The TI-ASC is a 650K 32 bit word machine with IBM 360-like floating point and vector and scalar hardware. It has 48 registers.

Texas Instruments 9900/4 (Vienna)


2. MACHINE. TI 9900/4.

3. SYSTEM CONFIGURATION. No operating system; requires a mark-sense card reader and a line printer.

4. DOCUMENTATION. The system (hardware and software) is sold for 200,000. Austrian Shillinga (about $1500 U.S.).

5. MAINTENANCE. Available in the form of a supplement to the Pascal Report. (* Not known if this is machine retrievable. *)

6. MAINTENANCE. "We intend to make more of it [the system] and we would like to accept bug reports."

7. STANDARD. The following are not supported: files; with and goto; formal procedures/functions. Sets of 64 characters are supported.

8. MEASUREMENTS. It is very slow compared with other systems. The system uses 12K ROM words and no external memory.

9. RELIABILITY. The reliability of the system is excellent. (* Date first released and number of sites using system not reported. *)

10. DEVELOPMENT METHOD. The system is written in Pascal and machine code (300 source lines). It took 3 months to implement it on any microprocessor with no special experience of the implementors. The machine independent parts are bootstrapped by an existing Pascal compiler. The system is intended primarily 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 leaf 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. The interpreter performs all context-sensitive checking at execution time.

Machine independent parts of the system, i.e., the compiler and part of the interpreter are in the intermediate language. Only the nucleus of the interpreter is machine-dependent and therefore hand-coded. The input device is a mark-sense card reader accepting specially coded cards (reserved words have their own punch codes).

11. LIBRARY SUPPORT. (* No information *)

Univac 90/30.

See letter from C.G. Handley under Hewlett Packard HP-21XX.

Univac 90/70.

See Siemens 4004, 7000 series.

The U9070 (formerly RCA Spectra 70) is very similar to the Siemens machines, both in hardware and software (WMS vs RS2000).

Univac 1100 (San Diego)

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Michael S. Ball, Code 632, Naval Ocean Systems Center, San Diego, CA 92152. (* No phone number provided. *)

2. MACHINE. Univac 1100 series.

3. SYSTEM CONFIGURATION. Exec-8 operating system; can be run in Demand mode.

4. DISTRIBUTION. As a member of UNIX, you may request a copy from Mike by sending a mag tape and noting any restrictions on its format.

5. DOCUMENTATION. A 29 page machine readable supplement to the Pascal User Manual and Report entitled "Pascal 1100" documents the implementation.

6. MAINTENANCE. (* No information *)

7. STANDARD. Restrictions: entry, processor, and univ are reserved words; standard procedures and functions may not be passed as actual parameters; file of file is not allowed. Sets may have at most 144 elements. The compiler accepts the full ASCII character set. A compiler option allows processing of Brinch Hansen Sequential Pascal programs.

8. MEASUREMENTS. The compiler compiles into 34K words and requires 6K words of library routines. Self-compilation requires about 15.5K for stack and heap.

Execution times for code compiled by Pascal was compared with code generated by the NUALG and ASCII FORTRAN processors. Fortran's local optimization was taken as a base value. The programs used for comparison were taken from Wirth's paper on the design of a Pascal compiler (Software - Practice and Experience, Vol. 1 (1971), pages 309-333). The results are summarized in the following table.

<table>
<thead>
<tr>
<th></th>
<th>Pascal no tests</th>
<th>NUALG no tests</th>
<th>FORTRAN (local opt. global opt.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PART</td>
<td>0.62</td>
<td>0.61</td>
<td>0.94</td>
</tr>
<tr>
<td>PARTINP</td>
<td>1.18</td>
<td>1.06</td>
<td>1.37</td>
</tr>
<tr>
<td>SORT</td>
<td>1.37</td>
<td>1.12</td>
<td>1.49</td>
</tr>
<tr>
<td>MATH</td>
<td>1.82</td>
<td>2.03</td>
<td>2.03</td>
</tr>
<tr>
<td>COUNT</td>
<td>0.30</td>
<td>0.72</td>
<td>0.66</td>
</tr>
</tbody>
</table>

SEPTEMBER 1977
9. RELIABILITY. Quite good; it should approach excellent. The system has been in local use since about February, 1976, and it has been installed at 25 sites (11 university, 4 government, 1 industry).

10. DEVELOPMENT METHOD. The compiler was developed from Pascal-P2. (*) Person-hours to develop system not reported. *)

11. LIBRARY SUPPORT. Generated code can be linked to subprograms written in Fortran or assembler.

Univac 1100 (Madison)

ACADEMIC COMPUTING CENTER
THE UNIVERSITY OF WISCONSIN - MADISON
1210 WEST DAYTON STREET
MADISON, WISCONSIN 53706
608-262-1105
August 31, 1977

PASCAL Implementations
University Computer Center
227 Experimental Engineering Bldg.
University of Minnesota
Minneapolis, MN 55455

Dear Mr. Bonham:

Enclosed please find a description of our new diagnostic PASCAL compiler. The following will outline the development of the compiler (which isn't specifically dealt with in the description).

The UW-PASCAL compiler is the joint effort of five people (myself, Richard LeBlanc, Masahiro Honda, Steve Zeigler and Gary Holmes). It currently represents about 24-30 man months. Design was initiated during the summer of 1975 and the first test version was released to users in late 1976.

The compiler was designed from scratch, using a syntax-directed organization. The compiler uses a table-driven LALR(1) parser and an error corrector which is driven by the parsing tables. Initially the compiler was bootstrapped through a version of the P-compiler. Later, Mike Ball's N.D.S.C. compiler was used. At present, of course, we bootstrap through our own compiler. This has the added benefit of allowing our diagnostic checks to monitor our own compiler (at a very acceptable level of overhead). Indeed, the preponderance of compiler bugs are found in this manner. As a result, errors are automatically linked to the offending source statement in the compiler and readily fixed.

In case you are interested, I'm including a copy of our current User's guide (an updated version is being prepared). I'm also posting a copy of the compiler description to Andy Mickel for inclusion in the PUG Newsletter (or are you the person who handles that department?)

If you'd like further information, please feel free to write me.

Sincerely,

Charles N. Fischer

P.S. A description of our new diagnostic PASCAL compiler.

Sincerely,

Charles N. Fischer

P.S. A description of our new diagnostic PASCAL compiler.
Diagnostic PASCAL Compiler for Univac 1100 Series

The University of Wisconsin-Madison Academic Computing Center (MACC) has developed a diagnostic PASCAL compiler for the Univac 1100 series. The compiler is especially designed for research and instructional use. It emphasizes careful and complete diagnostic checking at both compile-time and run-time. Included are subscript and subrange checks, pointer validity checks, record variant and set range checks. When run-time errors are discovered a procedure walk back (with source program line numbers) as well as a symbolic dump of scalar variables are available. During compilation, a complete analysis of the syntactic and semantic correctness of the source program is performed. Automatic correction of minor syntax errors (e.g., missing semicolons or parentheses) is included.

The following provides detailed information about the compiler and its distribution policy.

1. The UW-PASCAL compiler is an ASCII processor which operates on any Univac 1100 series computer under EXEC-8. Two versions of the compiler are available. The first produces standard relocatable elements which may be collected to produce executable absolute elements. The second version operates in a "Load and Go" manner. PASCAL source programs are compiled directly into core and immediately executed. No collection step is used.

2. UW-PASCAL is written in PASCAL. Its source (including all versions) is about 34K lines. Compilation speed is about 4000 lines/min (on an 1110). The compiler requires about 70K words to operate (which is larger than most other Univac compilers). However overall compilation costs appear to be comparable to other Univac ASCII compilers. Code generated by the compiler is as good as, or better, than that generated by other ASCII compilers operating in a non-optimizing mode.

   The Load and Go version is marginally smaller and faster than the standard compiler. For small programs, its cost (for compilation and execution) is about 60% of the standard compiler.

3. UW-PASCAL implements all of the Standard PASCAL language with the exception of GOTO's out of procedures. (STOP and ABORT statements are available as a partial alternative.)

   In addition to the extensive diagnostic capabilities noted above, a number of other language extensions are available. These include:

   (a) A very powerful external compilation capability. Procedures which are compiled independently are always compiled in the environment of their declaration. This allows complete compile-time checking of procedure interfaces as well as access to global variables. Linkage to externally defined assembly language procedures is also provided.

   (b) Conditional compilation facilities are provided. These include the optional compilation of code sequences enclosed by comment brackets ("conditional comments"). Further, code is not generated for unreachable program statements.

   (c) The DISPOSE procedure is implemented. Run-time pointer checks ensure that disposed objects cannot subsequently be referenced. Heap objects may be grouped into logical units ("sub-pools") which may be freed in a single DISPOSE operation. This often significantly simplifies reclamation of heap storage.

4. UW-PASCAL has been in use since early in 1977. It has received rather heavy use and has been found to be very reliable (at present no extant bugs are known).

   MACC currently maintains UW-PASCAL as a fully supported software product. The compiler, with a year of compiler support, is available for a fee of $750. Both the source and absolute modules of both versions of the compiler as well as PASCAL support routines will be provided. Prompt distribution of corrections to compiler bugs as well as improvements to the compiler are also included. After this initial period, continuing support (including compiler improvements and extensions) is available for a fee of $600 per year. A surcharge of $100 will be added for users outside of the United States.

   A UW-PASCAL User Guide (included in machine-readable form with the compiler) which further details this compiler is available for a postage and handling fee of $3 ($5 foreign). Normally, the compiler is distributed on a 9-track 1600 BPI tape. However, other tape formats and densities may be available upon special request.

   All inquiries should be directed to:

   PASCAL Development Group
   Attn: Dr. C. N. Fischer
   MACC
   1210 West Dayton Street
   Madison, Wisconsin 53706

   Information may also be obtained by contacting Dr. Charles N. Fischer at (608) 262-7870.

5. UW-PASCAL is an on-going research project at the University of Wisconsin. Future development plans include:

   (a) Compiler tuning to reduce core requirement (to about 45K for
small programs) and to reduce overall compilation costs.

(b) Inclusion of a varying-length string manipulation capability (similar to PL/1 varying length strings), with concatenation and substring operations, I/O, etc.

(c) Addition of an interface to ASCII Fortran subprograms.

Varian (Sperry-Univac) V-70

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Distributed by the Varian Users Group (VOICE), Varian Data Machines (Sperry Univac), 2722 Michelleon Drive, Irvine, CA 92713 (714/833-2400).

2. MACHINE. Varian V-70 series.

3. SYSTEM CONFIGURATION. Requires 32K+ memory, memory map, Vortex II operating system, extended instruction set, and 512 words of writable control store (WCS).

4. DISTRIBUTION. Available from Varian as VOICE #183CB.

5. DOCUMENTATION. A 120 page manual (non-machine retrievable) is available as part of distribution.

6. MAINTENANCE. (* no information *)

7. STANDARD. This is Brinch Hansen style Pascal. I/O is non-standard and oriented toward the Vortex-II I/O macro. Reference to files is by unit number. Additional restrictions: Strings must have an even number of characters. Goto's are not supported. Enumeration types cannot be defined within record declarations. Records may have at most 16 variants, and the ordinals of the variant labels (constants) must be in the subrange 0..15. Sets may have at most 128 elements. Uses Mark-Release.

8. MEASUREMENT. Compiles over 1000 statements per minute. Compiler requires 17K decimal words of main memory.

9. RELIABILITY. Good. Distributed to over 10 sites.

10. DEVELOPMENT METHOD. Based on Brinch Hansen's Sequential Pascal.

11. LIBRARY SUPPORT. (* no information *)

Xerox Sigma 6, 9.

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Pierre Desjardins, Universite de Montreal, Informatique, C.P. 6128, Montreal 101, Quebec, Canada (514/343-7662).

2. MACHINE. Xerox Sigma 6 and 9.

3. SYSTEM CONFIGURATION. (* no information *)

4. DISTRIBUTION. Distributed on magnetic tape (9 track, 800 bpi, structured much like a standard Xerox processor distribution tape - labelled in account SYSGEN). Distribution cost is $250, payable to Pierre Desjardins. The distribution includes documentation.

5. DOCUMENTATION. Program comments are in English. The following documents are distributed: "Program Description" (English) contains installation and maintenance information; "Manuel d'utilisation..." (French) is the user's manual; "METAPASC..." (French) provides macro-procedures to aid writing external procedures or functions in Meta-symbol; "Pascal 2 - Sigma: un systeme de programmation Pascal" (French) describes the functional structure of the compiler.

6. MAINTENANCE. Bug reports are welcome, and "update sheets could be sent." The distribution fee does not imply any responsibility or maintenance service on the part of the distributor, implementor, or the Universite de Montreal.

7. STANDARD. Corresponds to Pascal User Manual except: files may not be components of arrays, records or files; string constants may not occur in the const section; standard procedures and functions may not be passed as actual parameters. Sets may have at most 32 elements.

8. MEASUREMENT. Compiler peak code size is 25K. Self-compilation takes 35K. Compilation rates are: 600 lines per minute (Sigma 6 - BPM/STM) and 1200 lines per minute (Sigma 9 - CP=V). (* Size and execution speed of generated code not reported. *)

9. RELIABILITY. Good to excellent. (* Date first release and number of sites using system not reported. *)

10. DEVELOPMENT METHOD. The compiler source is 6200 lines of Pascal. It was produced by cross-compiling from a CUC Cyber 74. Effort was 18 person-months (without any prior knowledge of Sigma machines).

11. LIBRARY SUPPORT. The compiler produces a relocatable object module (in Xerox Standard Object Language) for each procedure and function. Provision is made for external procedures and functions written in Meta-symbol.

Xerox Sigma 7.

See also CII 10070.

The CII Iris-80 compiler (described above) has been transposed to the the Xerox Sigma 7 running under the BPM monitor by Masato Takechi, formerly at Department of Mathematical Engineering, University of Tokyo, Bunkyo-ku, Tokyo 113, Japan. Present address: Department of Computer Science, University of Electro-Communications, 1-5-1 Chofugaoka, Chofu-shi Tokyo 182, Japan.

Another Sigma 7 project, apparently incomplete and inactive, was carried out by Henry Bauer, III, Computer Science Department, University of Wyoming, Box 3682, Laramie, WY 82071 (307/766-5134).

21log 2-80.

Ken Bowles and co-workers, UCSO, have adapted the San Diego DEC LSI-11 Implementation to run on the 21log 2-80 running (at 2.5 MHz) about 70% as fast as the LSI-11. Release is expected by the end of 1977. See the DEC LSI-11 (San Diego) note, above.