

Pascal News

(FORMERLY PASCAL NEWSLETTER)

NUMBER 12

COMMUNICATIONS ABOUT THE PROGRAMMING LANGUAGE PASCAL BY PASCAL

JUNE, 1978

TABLE OF CONTENTS

COVER: The PUG Letter Opener and Letter-Writing Implement

0	POLICY: <u>Pascal News</u>
1	ALL PURPOSE COUPON
3	EDITOR'S CONTRIBUTION
4	HERE AND THERE WITH PASCAL
4	News (Jobs, Tidbits)
7	French/English - English/French Pascal Identifiers
8	Pascal in the News
8	Conferences
9	Books and Articles
11	Articles Wanted
11	Roster Increment
17	APPLICATIONS
17	News
18	Algorithms
20	Software Tools
32	Programs
32	ARTICLES
32	"Extensions to Pascal for Separate Compilation" - Richard J. LeBlanc
33	"What Are Pascal's Design Goals?" - Robert D. Vavra
34	"Pascal Environment Interface" - Terje Noodt
37	"Subranges and Conditional Loop" - Judy M. Bishop
39	"A Few Proposed Deletions" - John Nagle
40	OPEN FORUM FOR MEMBERS
52	Pascal Standards
56	IMPLEMENTATION NOTES
56	Checklist
56	Portable Pascals
57	Feature Implementation Notes
57	Machine-Dependent Implementations
68	Index to Implementation Notes (PUGN 9-12)
69	POLICY: Pascal User's Group

POLICY: PASCAL USER'S GROUP (78/04/15)

Purposes: Pascal User's Group (PUG) tries to promote the use of the programming language Pascal as well as the ideas behind Pascal. PUG members help out by sending information to Pascal News, the most important of which is about implementations (out of the necessity to spread the use of Pascal).

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

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

See the ALL PURPOSE COUPON for details.

FACTS ABOUT Pascal, THE PROGRAMMING LANGUAGE:

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

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

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

Pascal is a leading language in computer science today and is being used increasingly in the world's computing industry to save energy and resources and increase productivity.

Pascal implementations exist for more than 62 different computer systems, and the number increases every month. The Implementation Notes section of Pascal News describes how to obtain them.

The standard reference and tutorial manual for Pascal is:

Pascal - User Manual and Report (Second, study edition)

by Kathleen Jensen and Niklaus Wirth

Springer-Verlag Publishers: New York, Heidelberg, Berlin

1978 (corrected printing), 167 pages, paperback, \$6.90.

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

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

Pascal User's Group is each individual member's group. We currently have more than 1923 active members in more than 35 countries. This year Pascal News is averaging more than 150 pages per issue.

PASCAL USER'S GROUP

USER'S

GROUP

ALL PURPOSE COUPON

(78/04/15) •

Pascal User's Group, c/o Andy Mickel
University Computer Center: 227 EX
208 SE Union Street
University of Minnesota
Minneapolis, MN 55455 USA

+ Clip, photocopy, or
+
+ reproduce, etc. and
+
+ mail to this address.

// Please enter me as a new member of the PASCAL USER'S GROUP for ___ Academic year(s) ending June 30, _____ (not past 1982). I shall receive all the issues of Pascal News for each year. Enclosed please find _____. (* Please see the POLICY section on the reverse side for prices and if you are joining from overseas, check for a PUG "regional representative." *)

// Please renew my membership in PASCAL USER'S GROUP for ___ Academic year(s) ending June 30, _____ (not past 1982). Enclosed please find _____.

// 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} _{phone} 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} _{phone}. 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.
- please enclose the proper prepayment (checks payable to "Pascal User's Group"); 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.
- please remember that PUG is run by volunteers who don't consider themselves in the "publishing business." We produce Pascal News as a means toward the end of promoting Pascal and communicating news of events surrounding Pascal to persons interested in Pascal. We are simply interested in the news ourselves and prefer to share it through Pascal News, rather than having to answer individually every letter and phone call. We desire to minimize paperwork, because we have other work to do.
- American Region (North and South America): Join through PUG(USA). Send \$6.00 per year to the address on the reverse side. International telephone: 1-612-376-7290.
- European Region (Europe, North Africa, Western and Central Asia): Join through PUG(UK). Send £4.00 per year to: Pascal Users' Group/ c/o Computer Studies Group/ Mathematics Department/ The University/ Southampton SO9 5NH/ United Kingdom. International telephone: 44-703-559122 x700.
- Australasian Region (Australia, East Asia -incl. Japan): Join through PUG(AUS). Send \$A8.00 per year to: Pascal Users Group/ c/o Arthur Sale/ Dept. of Information Science/ University of Tasmania/ Box 252C GPO/ Hobart, Tasmania 7001/ Australia. International Telephone: 61-02-23 0561.

PUG(USA) produces Pascal News and keeps all mailing addresses on a common list.

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

RENEWING? (Costs the same as joining.)

- 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. Renewing for more than one year saves us time.

ORDERING BACKISSUES 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) are out of print.
(A few copies of issue 8 remain at PUG(UK) available for £2 each.)
- Issues 9, 10, 11, and 12 (September, 1977 - June, 1978) are available from PUG(USA) all for \$10 and from PUG(AUS) all for \$A10.
- extra single copies of new issues (current academic year) are:
\$3 each - PUG(USA); £2 each - PUG(UK); and \$A3 each - PUG(AUS).

SENDING MATERIAL FOR PUBLICATION?

- check the addresses for specific editors in Pascal News. Your experiences with Pascal (teaching and otherwise), ideas, letters, opinions, notices, news, articles, conference announcements, reports, implementation information, applications, etc. are welcome. "All The News That Fits, We Print." Please send material single-spaced and in camera-ready (use a dark ribbon and lines 18.5 cm wide) form.
- remember: All letters to us will be printed unless they contain a request to the contrary.

MISCELLANEOUS INQUIRIES?

- Please remember that we will use Pascal News as the medium to answer all inquiries, and we regret to be unable to answer individual requests.

POLICY: PASCAL NEWS (78/04/15)

- * 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 3 or 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 material (brevity is a virtue) for Pascal News single-spaced and camera-ready (use dark ribbon and 18.5 cm lines!).
- * Remember: ALL LETTERS TO US WILL BE PRINTED UNLESS THEY CONTAIN A REQUEST TO THE CONTRARY.
- * Pascal News is divided into flexible sections:

POLICY - 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 in the news, history, membership rosters, etc.

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

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

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

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

- * Volunteer editors are (addresses in the respective sections of Pascal News):

Andy Mickel - editor

Jim Miner and Tim Bonham - Implementation Notes editors

Sara Graffunder - Here and There editor

Rich Stevens - Books and Articles editor

Rich Cichelli - Applications editor

Tony Addyman - Standards editor

Scott Bertilson, John Easton, and Steve Riesman - Tasks editors



UNIVERSITY OF MINNESOTA
TWIN CITIES

University Computer Center
227 Experimental Engineering Building
Minneapolis, Minnesota 55455

(612) 376-7290

The DEADLINE for PUGN 13/14 is August 15. Tony Addyman is now PUG's new Standards Editor. Don't forget to renew if you need to--check your mailing label.

Personal Observations

- 1) Pascal-P has enabled a great many people to learn about compilers who otherwise would never have had the chance. Do you realize the implications? These same people (myself included) will never be able to look at other compilers for other languages (especially the ones peddled by manufacturers) the same way from now on. Our critical eyes probably won't be able to endure them either.
- 2) Please see the Books and Articles section for the article entitled: "Ambiguities and Insecurities in Pascal," which is the first, good, critical article about Pascal to appear (yes, we know about Habermann's article). The most memorable passage is in the conclusion:
"...Because of the very success of Pascal, which greatly exceeded the expectations of its author, the standards by which we judge such languages have also risen. It is grossly unfair to judge an engineering project by standards which have been proved attainable only by the success of the project itself, but in the interests of progress, such criticism must be made."
- 3) Many people are now decrying the lack in Pascal of "business-oriented" language features such as indexed-sequential access methods for file processing, packed decimal data types, and other inefficient ways of doing computing. I would suggest a Business Procedure Library similar to the IMSL and NAG mathematical and statistics libraries for numerical (old term = 'scientific') people. We should use the simple, but versatile tools (language features) we already have to build what we need for other things.
- 4) We need more news (notices, articles, opinions, etc.) for Pascal News about teaching experiences with Pascal.

How is Pascal User's Group? (*new members especially please read this*)

PUG has now grown too large to handle it in the personal manner we have in the past. Membership stands at 2147+. We used to be extremely efficient, because I, for one, could keep it all in my head and remember who was a member from where, which joined when and how. It was like stamp collecting. We have resorted to dropping all kinds of services we never promised to do but nevertheless did. Now when a new member joins, all he or she receives is backissues, and no personal reply, receipt, or answers to questions. PUG is another example illustrating limits to growth.

The event that seems to have changed the situation permanently was the first full-page article about Pascal in the April 27 issue of Computerworld--the largest and most widely-read computer journal in the United States. The following Monday we received 83 pieces of mail in one day (old record for a single day was 39 pieces, while typical mail in the past averaged 20-30 pieces/week.)! Do you realize how much time it takes to open 83 pieces of mail? Remember, we don't have secretaries.

PUG(USA) has managed to break even in the past--including this year--but we must raise the rates to \$6 per year. Postage and printing costs keep rising. David Barron at PUG(UK) announces new rates of 4 per year and Arthur Sale at PUG(AUS) announces a \$A2 decrease (now \$A8). Please see their notices following. At least now our rates are more normalized. We have kept the rate low to attract members and to spread Pascal as fast and as far as possible. We as a group are an exceptionally broad base of people, and I think that is a real accomplishment. And remember, we accept no advertising.

We have always tried to keep this operation simple: no special services, no special rates for special mailing, etc. I know I just wouldn't have time otherwise. I set up PUG so that it can be dismantled within one week and all money refunded! Charging a little more money this year will allow us to hire a part-time secretary to handle the growing clerical workload. The most time-consuming process is to process memberships and update the mailing list. We usually batch 3 or 4 weeks of mail before we process it!

- Andy

78/05/02.

Editor's Contribution

Dear Andy,

Here is our cost estimates for Australasian distribution for 1978/9. As you will see, I am recommending a lowering of the fee to \$A8.00. Last year's fee was based on estimates from our printery which in the even proved slightly high, and of course the amalgamation of issues 9 and 10 saved us postage. Consequently we have a small reserve, and I have been able to budget for exactly balancing costs with subs in 1978/79, carrying any inflation in postage and the costs of carrying stocks of back copies out of the reserve.

There may be some request for refunds from people who paid for two years. I'd rather not be involved in sending out cheques, and I suggest we treat this the same as with people who pay for two years in a price rise situation: we don't ask for more so we shouldn't give refunds.

Printing cost per issue \$1.00

Postage : Australia \$0.70
New Zealand \$1.20
Singapore \$2.00
averaged over subscribers \$1.00 approx.

Cost per issue \$2.00

Recommended subscription for 1978/79 = \$8.00 (Australian)

Yours sincerely,

Arthur Sale

A.H.J. Sale,
Department of Information Science.

PASCAL USERS GROUP - European Region Subscriptions 1978/79

We regret that steep increases in the cost of printing compel us to increase the subscription to £4 PER ANNUM.

We regret the increase, but even at this figure we shall only just break even. Without volunteer labour, charges would be much higher.

Please remember that cheques must be in sterling, drawn on a British (or Irish) bank Processing sterling cheques drawn on foreign banks, or non-sterling cheques is prohibitively expensive.

If you have a Post-Giro account, you can pay by direct transfer into our account number 28 513 4000.

RENEWALS take time, which is precious. Why not subscribe for two or more years?

(*****)
REMEMBER TO RENEW YOUR PUG
MEMBERSHIP. NOTE THAT THE
RATES HAVE CHANGED!
) *****

Here and There With Pascal

NEWS

PASCAL JOBS

(* PUG Member Jack Laffe has been keeping track of some of the jobs for Pascalers which have been advertised in recent months. We decided to publish it as one more indication of the currency of Pascal. This is not a "Help Wanted" section; in fact, these jobs may have been filled. We may continue publishing this section, when space permits, if someone like Jack will compile the list. *)

(* The first job was advertised in CACM in January. The others all appeared in Computer World on the date indicated with the job description. *)

Softech: compiler design.

Dunhill Personnel Inc.: 78/01/02

Modular Computer Systems: 78/01/23

National Cash Register: compiler design for Pascal-like language: 78/01/30

Timeshare: applications, systems: 78/02/06

Amdahl: systems programmer with Pascal experience: 78/02/27

California State Universities and Colleges: instructional consultants in Pascal: 78/03/06

GTE Sylvania: software engineers: 78/03/20

Houghton-Mifflin: Pascal programmers: 78/3

TIDBITS

Richard E. Adams, 239 Chatham Road, Columbus, OH 43214: "Did you hear that Burroughs was implementing Pascal on a microprocessor? (They were advertising for people in Computerworld." (* 78/02/09 *)

Wayne Andrews, Electronics Department, Weber State College, 3750 Harrison Blvd., Ogden, UT 84408: "We just put Pascal on our Dec-10 system and are trying to get going on the project." (* 78/03/06 *)

C. Bailey, Bailey and Associates, 1144 S. Atlanta, Tulsa, OK 74104: "I have an Altair with 32K memory, 2 Altair Floppy Discs and a Decwriter. So I am interested in Pascal as implemented on the Altair or Altair-like CPU. I am employed as a programmer/analyst for the Altair and HP and SG minis." (* 77/12/30 *)

Francis H. Beardon, Manager of Projects, Data Systems, Cincinnati Electronics, 2630 Glendale-Milford Road, Cincinnati, OH 45241: "We at Cincinnati Electronics Corporation are interested in Pascal as a possible standard programming language for our developed software systems because of its projected portability." (* 78/02/13 *)

David J. Bell, 609 Craig Ave., Campbell, CA 95008: "My personal system is a Processor Tech SOI-10, with externally expanded memory and I/O. I am interested in developing a Pascal translator for this computer, and for the HP2112 I use at work." (* 78/03/10 *)

Brad Blasing, 1308 Centennial Hall, Univ. of MN, Minneapolis, MN 55455: "We have implemented the Netherlands Pascal compiler on our 11/40 running UNIX. Runs fast for an interpreter. It's a good hybrid of the P2 and P4 compiler. Could use a bit more user-type documentation." (* 78/04/02 *)

William R. Blatchley, Measurement Systems Div., Siemens Corp., 3 Computer Drive, Cherry Hill, NJ 08002: "We are engaged in test equipment design and development for memory devices and have a possibly immediate need for a Pascal implementation on a PDP-11 for testing magnetic bubble memories." (* 78/04/12 *)

Damon Blom, 72 Sandburg Drive, Sacramento, CA 95819: "I am presently using Pascal on an IBM 370/168 computer using a Pascal compiler written in XPL. I will be getting shortly

the Pascal compiler written in Pascal developed at SLAC, Stanford University." (* 78/03/05 *)

Richard J. Cichelli, 901 Whittier Dr., Allentown, PA 18103: "Joseph Mezzaroba at Villanova has supervised two projects to implement Pascal-S on the 370. One using the AAEC Pascal compiler (Pascal-S in Pascal--takes 100 seconds to compile) and the IBM PL/1 compiler (Pascal-S in PL/1--takes 36 minutes to compile). Pascal-S in Pascal was 30 per cent smaller and ran five times faster." (* 78/04/14 *)

Roger Creamer, CTB/McGraw-Hill, Del Monte Research Park, Monterey, CA 93940: "Also, any specific information which you could provide on Pascal implementations for the IBM 370 and DEC PDP-11 would be much appreciated." (* 78/04/14 *)

Anthony Conti, Box 1201, Concord, NH 03301: "I am a user of a Data General Eclipse S200 minicomputer and am interested in running and maintaining Pascal on it." (* 78/01/12 *)

Jean-Louis Decoster, Lyss-Str. 21, CH-2560 Nidau, Switzerland: "Could you inform me too if a Pascal compiler is already implemented for "Could you inform me too if a Pascal compiler is already implemented for the Motorola 6800?" (* 78/03/15 *)

Alan Delwiche, Computer Programming Instructor, Leland High School, 6677 Camden Ave., San Jose, CA 95120: "Would you please send me any information regarding versions of Pascal for an 8080 or Z80 microprocessor. We have a 32K Cromemco with dual minifloppy drives." (* 78/02/08 *)

Shaun Devlin, 6854 Cedarbrook, Birmingham, MI 48010: "I would also appreciate it if you could direct me to anyone who has or is planning to implement Pascal on a Texas Instrument 990/9900 system." (* 78/01/05 *)

Bob Dietrich, M.S. 60-456, Tektronix, Inc., P.O. Box 500, Beaverton, OR 97077: "Am bringing up solo-concurrent Pascal under RSTS/E time sharing system (PDP-11). Also involved with Swedish and BSM Pascals for PDP-11." (* 78/03/08 *)

Robert Emerson, Honeywell Information Systems, 9555 S.E. 36th St., Mercer Island, WA 98040: "Another interest of mine is implementing a Pascal compiler on the Honeywell Level 6 mini computer. Any tips for compiler implementation would also be appreciated." (* 78/01/17 *)

Mel R. Fisher, Business Dept., Calvary Community Church, 1175 Hillsdale Ave., San Jose, CA 95118: "I am in the process of writing specialized programs for our church records, bookkeeping, and data of this nature. The Pascal language sounds very interesting, and I would appreciate any further information that you could supply me with. We currently have an IMSAI 8080 48K memory, with floppy disk video display and printer." (* 78/02/15 *)

George H. Golden, Sr., Computer Center, SUNY-Fredonia, Fredonia, NY 14063: "We are trying to get Pascal running on the Burroughs B-4700. It runs. But takes too much core." (* 78/04/10 *)

Robert M. Green, Robelle Consulting, Ltd., No. 130, 10th Ave., Delta, BC V4M 3T9: "Could you let me know if there are any implementations of Pascal for the Hewlett-Packard 3000 computer? If not, I am interested in implementing it. Is there any way I can get a copy of the Portable Pascal compiler, version P4?" (* 78/02/02 *)

R. Gunzenhauser and R. Kleine-Homann, Institut für Informatik, Universität Stuttgart, 7 Stuttgart 1, Azenbergstr. 12, Germany: "We use Pascal as the first programming language for our freshman students and for high-school teachers.

"We offer Pascal at our German Computer TR 440; besides we have a DEC PDP 11/40 computer (OS DEC RSX 11-M, 92kBytes) and wish to implement Pascal or a Pascal subset like Pascal-S.

"We would be very obliged if you could send us information about Pascal implementations on RSX 11-M you know." (* 78/03/15 *)

Robert O. Harris, University College London, Computer Center, 19 Gordon Street, London WC1A 0AH, United Kingdom: "I read the bit on PUG finances and noticed that PUG (UK) were the big loss makers, so I reckon its time to stop reading the library copy and pay

for my own." (* 78/02/27 *)

Carroll Hennick, Autologic, Inc., 1050 Rancho Conejo Blvd., Newbury Park, CA 91320: "Your letter in SIGPC Notes was welcome."

Judy Herron, Computer Sciences Dept., Mt. San Antonio College, 1100 North Grand Avenue, Walnut, CA 91789: "In my recent reading, references to Pascal seem to pop up everywhere--although I have yet to see one line of source code. "I'm interested in learning what I can about the language, and its implementations. What manufacturers offer Pascal? Is there a compiler available for our Altair 8800, Xerox 530, or IBM 1130? It sounds as though Pascal is used mainly for the teaching of structured programming techniques. Are business and industry adopting it also?" (* 77/12/28 *)

Bruce Hillegass, Digital Equipment Corp., 146 Main St., Maynard, MA: "I obtained your name off a Pascal document located on one of our DEC Sys-10's. "Pascal is virtually unsupported on all of our in-house systems, and there are numerous versions of the compiler around. I have been interested in Pascal for quite a while, and I'm in the process of learning the language. I am exploring the possibility of writing a compiler using Pascal as the language and I'm looking into Pascal as a language used in micro-programming. "I would appreciate any information you may have on Pascal activities in university environments especially on the DEC Sys-10." (* 78/01/27 *)

Robert M. Hofkin, APIS Dept. C-014, Univ. Of CA-San Diego, La Jolla, CA 92093: "Language extensions seem necessary, but the syntax. Let's not have another PL/I! Also--why wasn't Cichelli's review of Ken Bowles' book critical? It sounded more like a product announcement from IBM." (* 78/03/17 *)

David Holland, P.O. Box 38243, Houston, TX 77088: "In case you don't know already, T.I. Are getting ready for a Pascal compiler on a ROM for their 16-bit TMS9900 MP." (* 78/01/31 *)

William F. Holmes, Washington University, School of Medicine, 660 South Euclid Ave., St. Louis, MO 63110. "We are not using Pascal at present, but are seriously considering it for the PDP-11 (including the LSI-11) and the 8080 or 6800. We also have Computer Automation's LSI-2's, but unfortunately do not use their operating system." (* 78/01/30 *)

William C. Hopkins, 1101 Bondsville Rd., Downingtown, PA 19335: ". . . still working on a Univac 90/70 implementation." (* 78/02/26 *)

Gary M. Huckabay, Department of Mathematics, Cameron University, Lawton, OK 73505: "I would appreciate information concerning the following: i) language definition, ii) implementation at any computer site, iii) any suggestions on implementation, iv) any information concerning implementation on the Hewlett-Packard 3000, Series II." (* 78/01/26 *)

Phil Hughes, P.O. Box 2847, Olympia, WA 98507: "I have been studying and debating whether to implement Pascal on a micro for over 6 months. The article 'Pascal vs. Basic' made me aware of two things: 1. There is a Pascal Newsletter. 2. I have been wasting my time thinking about what would make Basic better. "Please send me information on obtaining the Pascal Newsletter and any information you may have about implementations of Pascal on micros (particularly M6800's)." (* 78/01/19 *)

Joseph M. Jolda, Bartlett High School, Negus St., Webster, MA 01570: "I've been trying to build something around the IBM Assembler but I'm running into all sorts of problems. It seems as though Pascal has the possible answer for me." (* 78/01/09 *)

Ralph Johnson, 1592 N. Broad, Galesburg, IL 61401: "I am rewriting Concurrent Pascal for the PDP-11/40 which should take about two weeks. If no one else has done this, I will send you the few changes that need to be made to the PDP 11/45 version." (* 78/01/04 *)

Adnan Khan, 222/7, Block-E, (Opp. Walton Training Centre), Walton Road, Lahore, Cantt., Pakistan: "I would like to get some knowledge about the new developments made after my contribution of Source Library Mechanism for Pascal 1900, under George III, which has also reduced the compilation time by one third. My project also involved translation of some NAG routines into Pascal." (* 78/01/17 *)

James R. Kochanowicz, Dedicated Systems Inc., 180 N. Michigan Ave., Chicago, IL 60601: "We are presently using Pascal on a Sperry Univac V-76 series computer." (* 78/03/19 *)

Charles Kuhlman, New York City Criminal Justice Agency, 305 Broadway, New York, NY 10007: "We are preparing to gear up a DEC PDP 11/70 RSX-11P system and are contemplating use of Pascal for some applications. . . . Do you know specifically of any RSX 11/70 versions of Pascal?" (* 78/03/06 *)

Roland L. Lee, 645 35th Ave., San Francisco, CA 94121: "I am thinking of writing a compiler for the Z-80 and would like some information on existing resident Pascal compilers that you know of for the Z-80." (* 78/04/01 *)

Alan M. Lesgold, LRDC Computer Facility, University of Pittsburgh, 3939 O'Hara St., Pittsburgh, PA 15260: "I would be interested in knowing of sources, if they exist, for a 6800 cross-compiler that would run on a PDP-10 or PDP-15 and also for a PDP-15 compiler. I am very interested in implementing Pascal as our primary source language." (* 78/01/12 *)

Bruce MacAnespie, 600 N. Hickory Ave., Apt. 18, Bel Air, MD 21014: "If you can supply me with any contacts or information regarding Pascal compilers or interpreters implemented on Burroughs B6700 or B7700 Computer systems, please send it by return mail. Having been a Burroughs Algol fan for some years, I am extremely interested in a language that promises to be the next generation of decent software implementation languages." (* 78/03/08 *)

Mario Magidin, Direccion General de Sistemas y Procesos Electronicos, Subdireccion de Sistemas "B," Corregidora No.8, Centro, Palacio Nacional, Mexico 1, D. F.: "We are the computing facility of the Mexican Ministry of Budget and Planning. With the aid of a CDC Cyber-173 we are supposed to satisfy all the computing requirements of the Ministry, thus, large, so-called commercial type systems are constantly under development and/or running at our place.

"Up to now, all these systems have been programmed in COBOL, and although we are painfully aware of the shortcomings of this approach, (particularly with CDC's COBOL) our solutions were directed mainly towards the use of a preprocessor of the type of Weinberg's Metacobol.

"The idea of replacing COBOL with PASCAL has arisen. I would deeply appreciate your comments on this idea." (* 78/03/31 *)

Bill Marshall, Jr., Sanders Associates, Inc., 24 Simon St., Nashua, NH: "I've been praising and promoting Pascal for five years now . . . it's about time I put my money where my mouth is!"

Irv McKnight, 505 Cypress Point, No. 52, Mountain View, CA 94040: "I have an S-100 8080 system with a NorthStar Disc. Several of us are looking into making the U.C. San Diego Pascal system live in the NorthStar." (* 78/03/27 *)

Ronald D. McRaney, P.O. Box 10097, Station 1, Houma, Louisiana 70360: "I am in the process of putting together a Pascal dedicated PDP 11/03 for my personal use." (* 78/01/04 *)

J. Scott Merritt, 655 S. Fair Oaks Avenue, Apt. L-216, Sunnyvale, CA 94086: "Tried to find CACM article mentioned on Page 87 of PUG 11. It wasn't in Dec. '77 or anywhere else I looked. Where can I find it?" (* 78/03/11 We don't really know either; will you write to Amsterdam to ask? *)

Rolf Molich, Software Development Manager, Dansk Data Elektronik Aps., Generaltorvej 6A, DK-2730 Herlev: "Further, I would appreciate it very much if you could tell me the name and address of any person or institution that you may have heard of who is currently developing a Pascal compiler (not an interpreter) for the Intel 8080 microcomputer." (* 78/01/24 *)

Here and There With Pascal

Allan Moluf, 2317 Knob Hill, Apt. 9, Okemos, MI 48864: "I would like to suggest a new approach for Pascal compilers on small machines. Syntax table-directed parsing techniques are now getting acceptable error recovery and should result in much smaller compilers. If PUG members know of anyone working in this area, please suggest Pascal as a useful language to implement. Most of the code generation and library routines are available in a portable compiler, which should result in an easy project." (* 78/03/21 *)

Freeman L. Moore, Department of Computer Science, Pearce 203-B, Central Michigan University, Mount Pleasant, MI 48859: "For your records, CMU has a Univac 1106 computer with our version of Pascal from U.S. Naval Undersea Center, by M.S. Ball, version 1.1C4." (* 78/03/04 *)

Olav Naess, Welhavensgt. 65, Bergen Norway: "I am interested in a Pascal compiler for the Z-80 system I am building." (* 78/01/17 *)

Heidi L. Neubauer, Coordinated Sciences Lab, Univ. Of Illinois, Urbana, IL 61801: "I am using Pascal to write machine problems assigned in an operating systems course I am taking at the Univ. Of Illinois as a graduate student in Computer Science. Our class has used both standard Pascal and a souped-up version with concurrent processes and semaphores (still under development but workable)." (* 78/03/07 *)

William I. Nowicki, C.S.-R.L. Tech B626, 2145 Sheridan Road, Northwestern University, Evanston, IL 60201: "My special interest is the implementations of Pascal for mini-computers, especially PDP-8's and PDP 11's." (* 78/01/08 *)

David J. Pesec, 20130 Miller Avenue, Euclid, Ohio 44119: "I also am wondering if there is any copy of Pascal that will run on a Honeywell Series 60 processor." (* 78/01/30 *)

David Powers, 259A Trafalgar St., Petersham NSW 2049, Australia: "I have a TEC-9900 system (based on the TMS9900) on which I hope to eventually be able to use Pascal. I would therefore ask if you are able to assist in this--do you know of a Pascal compiler for the 9900, or of any way I could get (with a view to modifying for use with my system) the Pascal source for a compiler with a code generator for the PDP-11. . . or one of the other micros.

"I have been working on an implementation of Pascal-S for the 6502 (using 4-byte words) in the form of a cross-compiler (based on the compiler part of the Wirth Pascal-S interpreter as implemented in Pascal) to an 'ICODE' which runs on an interpreter (only partially debugged, as yet, being a translation of Wirth's 'interpret' procedure) running in 4K (5K-6K with floating point) using the Jolt 'DEMON' monitor. Are you aware of any similar implementations having been undertaken? Has anyone done, to your knowledge, the apparently feasible, but rather time-consuming conversion of this compiler into Pascal-S?"

Steven R. Rakitin, Combustion Engineering, Inc., Mail Stop 9488-4BB, 1000 Prospect Hill Road, Windsor, CT 06095: ". . . I am interested in the potential use of Pascal as a Process Design Language." (* 78/01/24 *)

Mike Rebmann, Memorex Corp., Communications Div., San Tomas at Central Expressway, Santa Clara, CA 95052: "We are potentially interested in adopting Pascal as a replacement for assembly language for programming our 1380 front end communications processor. Does the User's Group have any information on adopting Pascal for this purpose? I would be especially interested in the following kinds of stuff: 1. Compiler development (cost, time, feasibility of using 'weird' hardware features), 2. Cutting over a software development group to use the language (planning training, phasing). 3. Compatibility with existing software--it would be very hard to justify rewriting our existing product line software. 4. Support software development--library system, loaders, etc." (* 78/03/03 *)

D. Roberts, Computing Laboratory, University College of North Wales, Dean Street, Bangor, Gwynedd LL57 1UT, Wales, UK: "We have recently put H.H. Nagel's implementation of Pascal on our DECsystem 10." (* 78/03/17 *)

James D. Rogan, Comshare, Inc., P.O. Box 1588, Ann Arbor, MI 48106: "I have . . . included some documentation on the Pascal compiler implemented on our

company's computers. The use of the language is primarily for application production systems software. To date, COMSHARE has written marketable products in Pascal and we can currently cross-compile source for the Sigma 9 and an INTEL 8080 machine." (* 78/02/16 *)

Jon D. Roland, Computer Retailers Assn., Micro Mart, 1015 Navarro, San Antonio, TX 78205: "We plan to support Pascal and extensions thereof extensively during the years ahead. We expect Pascal and APL to emerge as the leading higher-level languages, although Cobol will probably remain popular among many of our business customers." (* 78/03/28 *)

Richard Roth, 5 North Salem Road, Ridgefield, CT 06877: "I implemented P-2 stack machine on Micro-Data 810 (but never finished compiler) and would like to get Pascal running on 8080/Z80 system under my disk OS (an advanced TOPS-10-like operating system)." (* 78/02/01 *)

Beardsley Ruml, 2nd, 3045 Ordway Street, N.W., Washington, DC 20008: "I would like to participate in [an implementation on a Z-80/8080] if possible but, if not, certainly want to be one of the first users." (* 78/01/25 *)

Robert L. Schoenfeld, Rockefeller Univ., 1230 York Ave., New York, NY 10021: "Interested in Concurrent Pascal and Modula for laboratory applications." (* 78/03/23 *)

Mike Settle, ICP, 2925 Merrell Road, Dallas, TX 75229: "I am not presently a user, BUT I WANT TO BE. I am particularly interested in 8080 and Z-80 implementations. I sure would like to see Pascal replace BASIC in the personal and home computing environment." (* 78/02/24 *)

Al Shpuntoff, Morningside College, Sioux City, IA 51106: "I would be delighted to be able to teach some of our courses using the facilities of Pascal, but alas, we are still using an antique IBM 1130 computing system. Still, the widespread availability of Pascal Compilers for mini-computer systems raised hopes. A direct question to one of the participants in these conversations brought forth the suggestion that you would know of the existence of a Pascal Compiler for the 1130 if anyone would." (* 78/04/07 *)

Michael L. Sieman, 6103 Harwood Ave., Oakland, CA 94618: "I would also be interested in knowing if the Pascal User's Group has available any other publications, particularly ones concerning the implementation of Pascal on small machines (I'm thinking especially of the DEC L-11 under the RT-11 system, or article indexes to past issues of the Pascal News (and are back issues available?)." (* 78/03/23 *)

George A. R. Silver, Earlham College, Richmond, IN 47374: "I am particularly interested in any recent issues which have reviews of implementations of Pascal on PDP 11/70's."

Roger Sippl, 1806 Toyon Lane, Newport Beach, CA 92660: "I learned Pascal while a student at UC Berkeley on the many versions of the compiler on a PDP 11/70, while it was being written and debugged. Not the recommended way to learn a language, but it had its merit. "I am now working as a consultant in California with a special interest in medical computer applications."

James A. Stark, M.D., 485 34th St., Oakland, CA 94609: "My computer resources are: IBM 370/148 at Univ. Of Calif. At San Francisco (Medical School) that has a batch Pascal compiler. UNIX at U.C. Berkeley has just completed the installation of a new interactive version by Joy, Graham, and Haley (complete with manual). I have a home brew 8080 with floppy on which I hope to install UCSD's version and a 6502 presently sitting that will be used to interface my I/O Selectric if and when I get a missing board from Numan Computer Exchange." (* 78/03/28 *)

Quentin F. Stout, Dept. Of Mathematical Sciences, SUNY-Binghamton, Binghamton, NY 13901: "Finally, I would greatly appreciate it if you could tell me where I could obtain a Pascal compiler for an IBM 370/158 under VSI. We are an academic institution which cannot afford a large fee, so we would probably have to obtain it from another university." (* 78/03/17 *)

Jeff Stroomer, 224 Heritage Lane, Exton, PA 19341: "Do you (or any of your readers) know

of a way to get Pascal's IF-THEN-ELSE's into LL(1)? I already know how to monkey with with LL(1) tables to make the parser work the right way, but that's not what I'm interested in; I want a grammar that's truly LL(1)." (* 78/01/13 *)

Roy Touzeau, Computer Science Dept., Univ. Of Montana, Missoula, MT 59812: "We have a version of Pascal for the DEC-10 working on the DEC-20." (* 78/03/07 *)

Mike Travis, Interdata, Inc., 3080 Olcott St., Suite 125A, Santa Clara, CA 95051: "I have just received the KSU version of Pascal which runs on an INTERDATA 8/32. We are now in the process of bringing it up in a multi-terminal environment in our local data center." (* 78/02/13 *)

Tim Walsh, 174 E. Maujer Street, Valley Stream, NY 11580: "I hope to implement a sub-set of Pascal on my 'KIU-1 expanded' sometime this year." (* 78/01/09 *)

Bill Winspur, Mgr., Computer Serv., Computer Dept. For Health Sciences, Univ. Of Manitoba, 753 McDermot Ave., Winnipeg, Manitoba R3E 0W3, Canada: "We are installing a CYBER 171 in March and plan to use Pascal on it. We are also getting into uProcessor applications and are particularly interested in a rumour of Pascal for the 8080." (* 78/02/03 *)

C. Dudley Warner, 16345 Los Gatos Blvd., No. 41, Los Gatos, CA 95030: "I have Z80 based uC w/64K mem etc.--running Pascal under CP/M and USCD 'Pascal.'" (* 78/03/08 *)

Anna Watson, 3705 Delwood Drive, Panama City, FL 32407: "My objective is to determine rather quickly whether we should specify a Pascal compiler in a new computer specification for use by our present Algol users. Hopefully, study of a Pascal Primer plus the Pascal News can indicate if Pascal can serve our needs." (* 78/03/20 *)

Chip Weems, Dept. Of Computer Science, Oregon State Univ., Corvallis, OR 97331: "I enjoyed talking to Tim B[onham] at the W. Coast Comp. Faire. Tell him that I'm rewriting my Pascal summary card, and will send him a copy when it's finished." (* 78/03/28 *)

John Withrow, DEC, MR1-1/A86, 200 Forest St., Marlboro, MA 01752: "I'm using the Pascal compiler on the DECSYSTEMS (10 and 20) here at Maynard and Marlboro, MA; as well as implementing a Pascal (subset) compiler." (* 78/01/25 *)

Sandra Wright, Defence and Civil Inst. Of Environmental Medicine, P.O. Box 2000, Downsview, Ont. M3M 3B9, Canada: "We plan on implementing Pascal under UNIX and RT-11 early in 1978." (* 77/11/30 *)

FRENCH/ENGLISH -- ENGLISH/FRENCH PASCAL IDENTIFIERS

(* We received the following list of correspondences between French and English Pascal identifiers from Patrick Ward at the University of Montreal. He credits Olivier Lecarme and Pierre Desjardins with the original translation. Since we expect this to be used simply as a reference by those reading programs in the other language, we are omitting CDC-specific identifiers and those local to Montreal. We also have a list made by A. Tisserant at Nancy. His list is slightly different. We'd appreciate some clarification from the Sous-Groupe Pascal about what is standard for the French identifiers. *)

French	English	English	French
abs	abs	abs	abs
allera	goto	and	et
alors	then	arctan	arctan
arctan	arctan	array	tableau
arrondi	round	begin	debut
avec	with	boolean	booleen
bas	downto	case	cas
booleen	boolean	char	car
car	chr	chr	carac
carac	chr	const	const

carre	sqr	cos	cos
cas	case	dispose	rendre
const	const	div	div
cos	cos	do	faire
dans	in	downto	bas
de	of	else	sinon
debut	begin	end	fin
detasser	unpack	eof	fdf
div	div	eoln	fdln
ecrire	write	exp	exp
ecrireln	writeln	false	faux
ensemble	set	file	fichier
entier	integer	for	pour
entmax	maxint	forward	plusloin
entree	input	function	fonction
et	and	get	prendre
etiqu	label	goto	allera
exp	exp	if	si
faire	do	in	dans
faux	false	input	entree
fdf	eof	integer	entier
fdln	eoln	label	etiqu
fichier	file	ln	ln
fin	fin	maxint	entmax
fonction	function	mod	mod
haut	to	new	nouveau
impair	odd	nil	nil
jusque	until	not	non
lire	read	odd	impair
lireln	readln	of	de
ln	ln	or	ou
mettre	put	ord	ord
mod	mod	output	sortie
nil	nil	pack	tasser
non	not	packed	paquet
nouveau	new	page	page
ord	ord	pred	pred
ou	or	procedure	procedure
page	page	program	program
paquet	packed	put	mettre
plusloin	forward	read	lire
pour	for	readln	lireln
pred	pred	real	reel
prendre	get	record	struct
procedure	procedure	repeat	repete
programme	program	reset	relire
rac2	sqr	rewrite	recrire
recrire	rewrite	round	arrondi
reel	real	set	ensemble
relire	reset	sin	sin
rendre	dispose	succ	succ
repete	repeat	sqr	carre
si	if	sqrt	rac2
sin	sin	text	texte
sinon	else	then	alors
sortie	output	to	haut
struct	record	true	vrai
succ	succ	trunc	tronc
tableau	array	type	type
tantque	while	unpack	detasser
tasser	pack	until	jusque
text	text	var	var
tronc	trunc	while	tantque
type	type	with	avec
var	var	write	ecrire
vrai	true	writeln	ecrireln

PASCAL IN THE NEWS

Australian (* A national daily newspaper *), February, 1978: Article in the "Computers" section about the Australian Atomic Energy Commission's compiler for the IBM 360 and 370 systems.

Byte, April, 1978: A letter from Stephen Smith describing the status of his work on a Pascal compiler, based on a subset of Pascal, for microcomputers. He is now testing the parsing procedures on a DECsystem 10.

Computer Weekly, February 23, 1978: NCR, Dundee, Scotland, is beginning to design and implement a language based on Pascal.

Computerworld, March 20, 1978: Hewlett-Packard's new language for operating system implementation, Syspal, combines many features of Pascal, Modula, Euclid, and Concurrent Pascal.

Computerworld, April 24, 1978: Richard Cichelli describes the "revolutionary" growth in use of Pascal, this despite the resistance of mainframe and system vendors. A short history of Pascal and the extent of implementations is presented.

Computing, January 5, 1978: A letter to the editor from R. J. Allwood in response to David Barron's earlier article in Computing. Allwood announces his reasons for rejecting a changeover from FORTRAN to Pascal and states what a tempting new language would look like.

Computing Europe, March 16, 1978: David Barron notes the choice of Pascal as a base for the U.S. Department of Defense language IRONMAN.

DARCOM (U.S. Army Materiel Development and Readiness Command) sent letters to PUG members on February 1, 1978, asking for their responses to a series of questions about use and implementation of Pascal. Purportedly, DARCOM is selecting a standard system software programming language. (* DARCOM got your name by copying it from the published roster. PUG has a general policy of not releasing the roster in machine-retrievable form.*)

Data-Link (* published by ACM-Los Angeles *), February 1978: G. S. Khalsa, managing partner of the Pasadena Byte Shop, is reported to view Pascal as becoming the standard language for micro business systems.

Datamation, February, 1978: A short announcement about PUG and Pascal News with information about how to join appeared in the "Source Data" section.

Datamation, February, 1978: A proposed multiprocessor system for the U.S. Navy, constructed by Lawrence Livermore Laboratories in California, contains a Pascal compiler, developed under subcontract by the Computer Sciences Department of Stanford Univ.

Instruments and Control Systems, December 1977: A report of a Pascal compiler under development by Texas Instruments to meet Dept. Of Defense specifications. The article suggests that TI's Pascal could become a de facto standard for minis and micros. Unlike Intel's PL/M, Pascal is not a proprietary language.

Journal of the Hewlett-Packard General Systems Users Group, January/February 1978: A short article introducing Pascal and some of its features and containing information about how to join PUG.

Mini-Computer News, April 27, 1978: A new Pascal software package for the DS990 packaged disk systems is announced by Texas Instruments. TI suggests that its Pascal, closely compatible with standard Pascal, has many applications in areas traditionally dominated by FORTRAN and COBOL.

PATCH (Univ. Of Notre Dame Computing Center's newsletter), March 1978: UND has recently installed Pascal.

UMMUG (Publication of the Univ. Of Minnesota Microcomputer Users Group), The University of Minnesota's recent acquisition of Terak computers and with them UCSD's Pascal compiler/interpreter is discussed.

Vogelback Computing Center Newsletter (Northwestern Univ.), April, 1978: In announcing a short course on Pascal, an article mentions the widespread acceptance of Pascal.

CONFERENCES

Australian Universities Computer Science Seminar, held February 23-24, 1978, University of New South Wales:

(* We received a letter from Tony Gerber saying that "everyone (Carroll (* Morgan *), Ken Robinson, Arthur Sale, Jeff Tobias, & Gordon Cox from AAEC, myself) was there." In addition, Tony sent us copies of two papers read at the conference:

G. W. Cox and J. M. Tobias, "An Implementation of Pascal for International Business Machines or The Impossible Takes a Little Longer."

(* From the abstract *) The programming language Pascal has successfully implemented for IBM360 and IBM370 computers under the OS/360 family of operating systems. The compiler is written in Pascal and fully supports Standard Pascal with some significant extensions. Interesting aspects of the relationship between the language and the IBM360 architecture are discussed. Surprisingly enough, the IBM360/370 general purpose architecture readily lends itself to an efficient implementation of a high-level language such as Pascal, although some features are impossible to realize.

Experiences in attempting to encourage a body of scientists to use Pascal in preference to FORTRAN are drawn on, with the conclusion that until a revised standard for Pascal is achieved, Pascal will never become a universally used programming tool.

Arthur Sale, "Mismatches and Conflicts Arising out of the Burroughs B6700/B7700 MCP and a Pascal Implementation."

(* From the abstract *) This paper draws on experiences of implementing a Pascal compiler on a Burroughs B6700 computer. Since these machines are designed for high-level language programming solely, and the operating system (MCP) is highly structured, the conflicts between the assumptions commonly made by Pascal adherents, or built into the language, and the facilities offered by the operating system posed some interesting conflicts which are examined herein.

Universite de Nice, Informatique, Mathematiques et Automatique, Manifestations Informatiques de Juin 1978, conference to include a meeting of the Pascal sub-group on the 13th and 14th of June.

(* Sorry we didn't know about this conference in time for the last issue. We'll hope to have titles of the papers presented by next time. *)

Second West Coast Computer Faire, March 3-5, 1978, San Jose, California. (* Pascal News editor Tim Bonham attended. He collected a dozen PUG memberships and reported that he "could have sold 100 Pascal User Manuals and Reports on the floor for twice their price." He also saw several demonstrations of Pascal on micros. Several papers of special interest to Pascalers are part of the proceedings *):

Sassan Hazeghi and Lichen Wang, "A Short Note on High Level Languages and Microprocessors."

(* From the abstract *) In this note, some of the practical aspects of bridging the gap between high level programming language and computer hardware are discussed. Several possible strategies are considered and the method of half-compiling-half-interpreting

is studied. In dealing with address space limitation (or tight memory situation) and slow speed of micro processors running an interpreter, a measurement and analysis technique is suggested. This analysis not only gives a good estimate of the timing and storage requirement before the actual implementation, it also helps to optimize the speed and storage usage of the implementation. The note concludes with some results concerning the implementation of the programming language Pascal on a family of micro-processors.

H. Marc Lewis, "An Experimental Pascal-like Language for Microprocessors.

(* From the abstract *) This paper describes an experimental Pascal-like high level language oriented to microprocessor implementation and use. The design criteria include modest memory requirements, self-compilation, simplicity, reasonable access to hardware features, and ease of extensibility. Program structure, data declarations, and control structures are described and examples given. Novel features of the language are discussed. An appendix gives a formal description of the language via syntax graphs.

Chip Weems, "An Introduction to Programming in Pascal."

(* From the abstract *) This paper will concentrate heavily on the use of the Pascal language at the beginner's level. A minimal knowledge of some other programming language such as FORTRAN, BASIC, or ALGOL is assumed.

The areas which will be covered are simple and structured statements in Pascal, simple and structured data types, plus procedures and functions. Emphasis will be placed on using Pascal statements, although some discussion of the power of user defined data types will also be included.

A list of machine models for which implementations of Pascal are known to exist is provided as an appendix.

BOOKS AND ARTICLES

PLEASE SUBMIT ALL NOTICES OF Pascal

BOOKS, ARTICLES, ABSTRACTS, etc.

to Rich for this section. Thanks, Andy.

Editor: Rich Stevens

Kitt Peak Nat'l Observatory
P. O. BOX 26732
Tucson, AZ 85726 USA

(phone: 1-602-327-5511)

APPLICATIONS

Patricia R. Mohilner, "Using Pascal in a FORTRAN Environment," Software Practice and Experience, 7:3 (June-July 1977), 357-362.

(* Summary of a review by R.A. Jones in Computing Reviews, January 1978. *) Mohilner demonstrates some problems encountered in attempting to write graphical applications programs in Pascal when the existing library of plotting routines was written in FORTRAN. She shows the solutions to those problems, but the example suggests that the problems she describes will likely be encountered by all installations.

V.A. Nepomniashchy, and L.V. Chernobrod, "Automatic Program Verification," Problems of Programming, 1976, pp. 63-80.

(* From the English summary in the table of contents *) "Describing the preliminary version of the system for proving assertions about programs (SPRUT). The deduction system herein is Hoare's system for proving correctness of programs. The input is a Pascal program with assertions. The verification condition generator outputs the list of lemmas to be proved by other blocks of the system. In algebraic and logical reduction of expressions simplification strategies are used, including axioms and lists of subgoals."

Gary J. Nutt, "A Comparison of Pascal and FORTRAN as Introductory Programming Languages," SIGPLAN Notices, 13:2, February 1978, pp. 57-62.

(* From the abstract *) "The Department of Computer Science at the University of Colorado has recently made the transition from FORTRAN to Pascal (in introductory courses), and this paper offers and informal discussion of the experiences of one instructor during that change."

IMPLEMENTATIONS

Urs Ammann, "On Code Generation in a Pascal Compiler," Software Practice and Experience, 7:3 (June-July 1977), 391-423.

(* From the abstract, as reported in Computing Reviews, January 1978. *) "This report deals with code generation in a Pascal compiler. It gives insight into the run-time organization of data and the use of the hardware registers of the underlying machine (a CDC 6400). It is shown how the compiler maintains a description of the register contents and uses this description to generate efficient code. Several examples of compiled code are discussed."

Forest Baskett, "The Best Simple Code Generation Technique for WHILE, FOR, and DO Loops," SIGPLAN Notices, 13:4 (April 1978), pp. 31-32.

(* From the abstract *) "This code generation technique for WHILE, FOR, and DO loops is simple to implement and usually results in the best loop code in the absence of flow analysis. Also the technique makes it possible to move code from inner loops without doing flow analysis and without ever moving code from a less frequently executed block to a more frequently executed block."

Kenneth L. Bowles, "The USCSD Pascal Project," Educom, 13:1 (Spring, 1978), pp. 2-7.

(* From the summary *) "Small stand-alone microcomputers can serve as the basis for running a sophisticated general-purpose interactive software system capable of supporting CAI, word processing, data processing, and other interactive tasks in addition to development of the software itself. The project described in this article has implemented such a software system using the Pascal programming language. The system is designed to be nearly machine-independent, and currently runs on a number of microprocessors, including the popular LSI-11, 8080, and Z80."

G. W. Cox and J. M. Tobias, "An Implementation of Pascal for International Business Machines or the Impossible Takes a Little Longer." (* See CONFERENCES section *)

Sassan Hazeghi and Lichen Wang, "A Short Note on High Level Languages and Microprocessors." (* See CONFERENCES section *)

H. Marc Lewis, "An Experimental Pascal-like Language for Microprocessors." (* See CONFERENCES section *)

Arthur Sale, "Mismatches and Conflicts Arising out of the Burroughs B6700/B7700 MCP and a Pascal Implementation." (* See CONFERENCES section *)

J. Welsh, "Economic Range Checks in Pascal," Software--Practice and Experience, Vol. 8 (1978), 85-97.

(* From the abstract *) "A Pascal implementation is described which exploits the information provided by subrange type declarations to minimize the run-time checking involved in detecting range violations. An evaluation of its performance is given, and some possible modifications are discussed. (* It pays to use sub-ranges. *)

LANGUAGES

Borge Christensen, "COMAL: Structured Basic," People's Computers, 6:4 (Jan.-Feb. 1978), pp. 36-41.

(* From the table of contents *) ". . . adding Pascal's algorithmic structures to BASIC."

M. Iglewski, J. Madey, and S. Matwin, "A Contribution to the Improvement of Pascal," SIGPLAN Notices, 13:1 (January, 1978), pp. 48-58.

(* From the introduction *) "The purpose of this paper is twofold. First of all we would like to present some of our proposals, concerning the desirable corrections in the Revised Report on Pascal and possible slight extensions of the language. Secondly we want to argue with some of the critical remarks on Pascal as formulated several months ago by Conradi."

Charles Lakos and Arthur Sale, "Is Disciplined Programming Transferable, and is it Insightful?" (* Received in January; may be published by now; more news from Arthur Sale or from PN 13 *)

(* From the abstract *) ". . . The paper applies the thought processes advocated by [E.W.] Dijkstra to [two] problems and indicates the insights that the authors gained from this. In both cases algorithms new to the authors were derived, and the properties of these are also examined. The paper . . . demonstrates that the techniques advocated by Dijkstra are indeed transferable to other programmers, and that this transfer yields better insight into the activity we call programming."

David Mundie, "Pascal vs. BASIC," People's Computers, 6:4 (Jan.-Feb. 1978), pp. 41-47. (* From the table of contents *) "A polemical comparison of the two as general-purpose microprocessor languages."

Jim des Rivieres and Ted Venema, "Euclid and Pascal," SIGPLAN Notices, 13:3 (March 1978), pp. 57-69. (* From the abstract *) "The programming language Euclid was intended for writing system programs that could be verifiable by state-of-the-art verification methods. Since verification was not an explicit goal in the design of Pascal, it is not surprising that this gave rise to differences between the two languages. The Euclid designers intended to change Pascal only where it fell short of this goal. This paper examines differences in the two languages in the light of this objective. These differences are roughly grouped under the headings verification, system programming, and user-oriented changes."

Abraham Silberschatz, Richard B. Kieburtz, and Arthur Bernstein, "Extending Concurrent Pascal to allow dynamic resource management," IEEE Transactions on Software Engineering, SE-3:3 (May 1977), 210-217.

(* One sentence from a review *) "The authors of this paper propose an extension to the programming language Concurrent Pascal to allow more flexible dynamic resource management. They introduce a new type called manager. . . ."

Tennet, R. D., "Language design methods based on semantic principles," Acta Informatica, 8:2 (1977), 97-112.

(* From the abstract *) "Two language design methods based on principles derived from the denotational 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 application of these principles, including a solution to the array parameter problem, and a modularization facility."

Arthur Sale, "Stylistics in Languages with Compound Statements" (* Article may have been published in Australia; check with Arthur Sale; more information in PN 13. *)

(* From the abstract *) "This short communication discusses a stylistic problem which arises in languages with use both statement separators such as semicolons, and begin-end bracketing structures, such as Pascal. It suggests that an alternative to the traditional rules which have evolved from Algol 60 is preferable."

Chip Weems, "An Introduction to Programming in Pascal." (* See CONFERENCES section. *)

J. Welsh, W. J. Sneeringer, and C.A. Hoare, "Ambiguities and Insecurities in Pascal," Software--Practice and Experience, Vol. 7 (1977), 685-696.

(* This is the best critical article to have appeared about Pascal. The ambiguities discussed are equivalence of types (name equivalence vs. Structural equivalence), scope rules and one-pass compilation, and set constructors. The authors point out the following "insecurities": features whose implementation either risks undetected violation of rules of the language or run-time checking that is too expensive to be tolerable: variant records, functions and procedures as parameters, range violations, uninitialized variables, and dangling references. *)

TEXTBOOKS

(* See reviews for Bowles, Conway, Grogono, and Schneider texts. *)

S. Alagic and M. A. Arbib, The Design of Well-Structured and Correct Programs, New York: Springer-Verlag, 1978, 292 pages, \$12.80. (* See description in No. 11. We hope to have a review in No. 13. *)

Richard Kieburtz (* Updated information *), Structured Programming and Problem Solving with Pascal, Englewood Cliffs, NJ: Prentice-Hall, 1977, 320 pages, \$12.80. (* Similar to Conway; see reviews. *)

A Guide to PASCAL Textbooks

Richard J. LeBlanc and John J. Goda
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA

G. Michael Schneider, Steven W. Weingart and David M. Perlman, An Introduction to Programming and Problem Solving with PASCAL, John Wiley, 1978. (\$12.95)

Among the strongest features of this book are its coverage of the complete programming processes (from problem specification through debugging and maintenance) and its emphasis on good programming style. Most of PASCAL is well presented, with the examples giving a good demonstration of how the features of the language should be used. The weakest part of the book is the presentation of "advanced" features such as variant records and pointers. Its coverage of programming fundamentals makes this an excellent text for an introductory course for students with little or no programming experience.

Peter Grogono, Programming in PASCAL, Addison-Wesley, 1978. (\$9.95)

While this is an introductory book, it concentrates more on the syntax of PASCAL and less on programming methodology than does the book by Schneider et. al. It is easy to read and has syntax charts integrated with the text rather than in an appendix. Grogono includes very good coverage of the advanced features, particularly pointers and dynamic data structures. The presentation of user-defined types is not as well-organized as it could be. While this book could be used as the text for an introductory course, its lack of coverage of programming fundamentals and its strength in the area of the advanced features make it best for students who have some programming experience.

Richard Conway, David Gries and E. Carl Zimmerman, A Primer on PASCAL, Winthrop, '76. (\$10.95)

This book is based on a PL/I book by Conway and Gries and it shows. The only structured type discussed is arrays and the discussion of user defined scalar types is very weak. The material on programming methodology is not at all integrated with the presentation of the language, so it is necessary to skip around in the book when it is being used as a text in an introductory course. There are errors in the presentation of PASCAL that are clearly not typographical. In general, this book fails to capture the idea that the features of PASCAL can actually make program development easier than if FORTRAN were being used.

Kenneth L. Bowles, (Microcomputer) Problem Solving Using PASCAL, Springer-Verlag, 1977. (\$10.95)

(See PASCAL News #11 for a more thorough review.) This basically appears to be a good book, but the language presented is not standard PASCAL. Bowles microcomputer PASCAL (with extensions for graphics and string handling) is used and the examples are heavily dependent on use of the extensions. This tends to make much of the book confusing to a student who does not have Bowles' system available.

ARTICLES WANTED

(* Tim Bonham has been surveying publications which might want articles about Pascal and has supplied us with the following list. *)

Wayne Green

Kilobaud

Peterborough, NH 03458

(603) 924-3873

Especially interested in articles which answer the questions "Why should I use Pascal?" "What uses does Pascal have in the real world?" and "How would I gain from using Pascal?" Maybe later on could use articles on how to program in Pascal. Will pay.

Stan M. Sokolow

Solus News

1690 Woodside Dr.

Redwood City, CA 94061

(415) 368-3331

Wants articles introducing and "justifying" Pascal. Especially interested in machines which are software compatible with the SOL (8080-based with cassette operating system).

Northstar Newsletter

2547 Ninth St.

Berkeley, CA 94710

Especially interested in software for the Northstar products: micro disk system and Horizon computer system. Introductions and justifications.

Larry Steckler, Editor

Radio-Electronics

200 Park Ave. S.

New York, NY 10003

(212) 777-6400

Introductions to programming in Pascal.

ROSTER INCREMENT (78/04/22)

The names listed below represent people who have renewed, changed address, or joined PUG since the roster increment was printed in PUGN 11. Some interesting statistics:

renew date	number	state	number	country	number
77	238	CA	349	USA	1535
78	1557	MN	156	UK	145
79	249	MA	107	Canada	88
80	54	TX	88	Germany	70
81	10	NY	83	Australia	42
82	7	PA	80	Netherlands	35
				Sweden	31
				Switzerland	27
				Denmark	23

RUSSEL J. ARBUTT 91330
 REESA ARRAMS 87196
 RICHARD E. ADAMS 43214
 TOMMY ADDICIAN H13 991 UNITED KINGDOM
 PETER ALTMANN D-5100 GERMANY
 J. AMBLER UNITED KINGDOM
 RICHARD H. AMEY 01752
 CHRISTOPHER AMLEY 55455
 DAVID E. ANDERSON 53201
 PETER ANDERSON 07102
 HANS APELKNANS S-350 03 SWEDEN
 RUBERTO ARGUETA EL SALVADOR
 MICHAEL C. ARISTONDI 94804
 WILLIAM J. ARTHUR 92624
 ANTTI ARVELA SF-33360 FINLAND

ATTENTION: GORDON R. SHERMAN 37946
 ATTENTION: G. TER HOFSTRAE MSV 259 CANADA
 ATTENTION: JERRY W. SEDGWICK 30332
 ATTENTION: R. D. BENGCHON 03824
 ATTENTION: SANDRA WRIGHT N31 309 CANADA
 ATTN: BETTE ROLLING-LIBRARIAN 45036
 ATTN: CECILIO CHILE
 ATTN: CENTRE DE RECHERCHE F-34075 FRANCE
 ATTN: CHICO PROJECT - CDC 95929
 ATTN: COMPUTER SCIENCE PRESS INC. 20854
 ATTN: CUPERINO LIBRARY 95014
 ATTN: DEPT DE MATHematIQUE AND INFORMATIQUE F-35031 FRANCE
 ATTN: DEPT OF BIOCHEMISTRY L61 781 UNITED KINGDOM
 ATTN: DEPT. OF COMPUTER SCIENCE 38677
 ATTN: DOCUMENTATION OFFICER NEW ZEALAND

ATTN: FREIE UNIVERSITAT BERLIN D-1000 GERMANY
 ATTN: FRIEDA S. COHN 53706
 ATTN: HELEN SMITH NZL 361 CANADA
 ATTN: INFORMATION SERVICE CENTER 55113
 ATTN: INSTITUT FUER INFORMATIK D-2000 GERMANY
 ATTN: INSTITUT FUR INFORMATIK D-2300 GERMANY
 ATTN: LAFAYETTE COLLEGE 18042
 ATTN: LIBRARY 91109
 ATTN: LIBRARY - COPY 2 94305
 ATTN: LIBRARY - COPY 2 97401
 ATTN: PROGRAMMING ADVISOR 89507
 ATTN: PROGRAMMING ADVISOR 89154
 ATTN: PURCHASING OFFICE 2600 AUSTRALIA
 ATTN: SAM CALVIN 09175
 ATTN: SCHOOL OF INFORMATION SCIENCES 2616 AUSTRALIA
 ATTN: SERIALS DEPT. 52242
 ATTN: SPECIAL INTERACTIVE COMPUTER LAB 55455
 ATTN: STUDENT COMPUTING COOP 92093
 ATTN: SUSAN BOWIE - DOCUMENTS ROOM 97403
 ATTN: THE LIBRARIAN H99 114 UNITED KINGDOM
 ATTN: UNBOUNDED COMPUTING 94086
 ATTN: UNIVERSITAT DE GENEVE F-38041 FRANCE
 ATTN: TECHNICAL RESEARCH CENTRE OF FINLAND SF-02150 FINLAND
 LEE D. AURICH 90230
 ROBERT M. BAER 94941
 C. SALLEY 74104
 F. T. BAKER 20760
 JOHN BANNING 94305
 ANH BARDIN 47401
 AHN V. BARROW UNITED KINGDOM
 FRANCIS H. BEARDEN 45241
 E. W. BEAUREGARD 17019
 GARY BECKWITH 55119
 H. D. BEER L69 30X UNITED KINGDOM
 DAVID A. BEERS 92701
 E. MAURICE BEESLEY 89557
 DAVID J. BELL 95008
 ABDUL RASAQ BELLO 55408
 STEVEN H. BELLOVIN 27514
 GILBERT BERGLASS 14221
 ALLEN BERGLUND 85005
 THOMAS BEYER D-2000 GERMANY
 JOEL BERSON 01821
 SCOTT S. BERTILSON 55455
 K. S. BHASKAR 68508
 LEONARD BINSTOCK 20901
 CHIKIS M. BISHOP NEW ZEALAND
 JUDY M. BISHOP 2001 SOUTH AFRICA
 K. W. BIXBY 92634
 ANTHONY BJEKSTEDT 74171
 R. S. T. BLACK D02 3XX UNITED KINGDOM
 ROBERT E. BLANTON 01178
 DAVID BLOM 95819
 W. ASHBY BOAZ 22101
 PETER ROCK 20052
 JOHN R. BOGAN 94010
 REFORO BOHD 73102
 RONALD V. BOSSLET 02154
 GREG BOURQUE 80901
 WILLIAM H. BRACK HONG KONG
 JOHN A. BRIGGS 84047
 JERRY R. BROOKSHIRE 77840
 DAVID H. BROHN 92017
 R. LEONARD BROHN 22903
 CHRIS BRYCE L85 4K1 CANADA
 MARTIN BUCHANAN 20007
 DONALD D. BURN 61801
 DEBORAH BUSCH 13346
 R. BUSH 97420
 RICKY W. BUTLER 23601
 JAMES D. CALLADINE LOC 1K0 CANADA
 T. M. CAMERON 32901
 JIM CAMERON 43023
 SHERRY L. CAMERON 92714
 ROBERT L. CANNON 29208
 ROY CARLSON 97077
 P. CARR BA2 7AY UNITED KINGDOM
 JOHN CARTER SR. 95050
 GEORGE P. CHENEY 01854
 C. W. CHILDERS 94550
 EARL N. CHRISTIANSEN 68134
 ROBERT G. CLARK FK9 4LA UNITED KINGDOM
 W. E. CLARK 92138
 JENNIFER CLARKE 02115
 JIM CLARKE 76010
 RICHARD CLAYTON H20 UNITED KINGDOM
 GEOFREY A. CLEAVE 3165 AUSTRALIA
 GERHARDT C. CLEMENTSON 80204

BETTY CLIFFORD T2N 1N4 CANADA
 D. S. COCHRANE UNITED KINGDOM
 JOE COLENTINE 75248
 R. COLE SE1 9AA UNITED KINGDOM
 HOWARD S. COLLEY JR. 95070
 TERENCE M. COLLIGAN 02193
 ANTHONY CONTI 03301
 MICHAEL J. COOK 95014
 D. P. COSTELLO 68583
 C. J. COTTON 19898
 H. MICHEL COURCHESNE H16 3S5 CANADA
 JIMM COUSINS 97005
 LAWRENCE F. CRAW 01701
 ROGER CREAMER 93940
 A. PAUL CROONBERGHUS 92277
 SCOTT CRUMPTON 32601
 HOWARD CUNNINGHAM 47904
 JOHN H. DALY 08108
 THOMAS DASHBURY 06880
 RONALD L. DANIELSON 95051
 GENE A. DAVENPORT 10016
 SCOTT T. DAVIDSON 92138
 ADELINO CARLOS DE SOUSA PORTUGAL
 DENNIS S. DELOUVE 55901
 ALAN DELMICHÉ 95120
 SHAWN DEVLIN 48010
 JERRY DI MASSA 95051
 ROBERTO DIAS BRAZIL
 DAVID R. DICK 02104
 CLEMENT L. DICKEY 93407
 MARY DIEGERT 13902
 BOB DIETRICH 97077
 H. JEAN-MARIE DIRAHD J1K 2R1 CANADA
 FELIX DREHER 66762
 LARRY DUBY 22209
 DOUGLAS DUNLOP 23185
 BRUCE J. EDMONDSON 94040
 JOHN EDWARDS 3083 AUSTRALIA
 P. S. EDWARDS 3216 AUSTRALIA
 H. W. EGDORF 80401
 JOSEPH EINBECK 88003
 JOHN ELDER B77 1NN UNITED KINGDOM
 LARRY E. ELLISON 08046
 ROBERT EMBERTSON 98040
 JAMES C. EMBERY 08540
 HANNU ERKIO SF-00100 FINLAND
 C. B. FALCONER 06504
 MICHAEL FAY 60919
 LINWOOD FERGUSON 22901
 BILL FINCH 95014
 MEL R. FISHER 95118
 P. G. FLITCHER 02154
 BOGG FORSTER 94611
 CRAIG FRANKLIN 03060
 R. D. FRENMAN UNITED KINGDOM
 ERIC D. FREY 03031
 DAVID P. FRICK 94025
 KARL FRYKELL 91125
 MICHEL GALLINIER F-31450 FRANCE
 DAVID GARDNER 55113
 PHILIP GAUDETTE 22301
 RICHARD D. GEORGE 60439
 CARSON GERMAN 90706
 BRANKO J. GEROVAC 02154
 CLIFFORD GERSTENHARER 55343
 CHARLES J. GIBBONS 68154
 RICK GILLIGAN 93407
 DON GILMORE 95030
 THOMAS GIVENTER 10583
 R. STEVEN GLANVILLE 94087
 GEORGE H. GOLDEN SR. 14063
 DAVIN GHABEL 02174
 TOM GRABOWSKI 15108
 JEFFREY W. GRAHAM 32223
 SUSAN L. GRAHAM 94720
 DAVID N. GRAY 78769
 ARTIE GREEN 19311
 ROBERT M. GREEN V4M 3T9 CANADA
 SY GREENFIELD 12206
 PETER GUGONO CANADA
 RICHARD D. HACKATHORN 19174
 WILLIAM L. HAFNER 19424
 JACQUES HAGUEL J1K 2R1 CANADA
 BYRON HALE 94043
 RICHARD W. HANLITON 94702
 BRIAN HANSEN 97213
 BILL HARGOOD 02154
 ANDY HARRINGTON 93017
 HIKE HARRIS 62704

ROBERT O. HARRIS MCIA OAH UNITED KINGDOM
ROBERT W. HARRIS 21114
DOM HARVEY 97079
W. F. HAYGOOD 84121
PETER HAYNES LSN 1K7 CANADA
SASSAN HAZEGHI 94305
RANDALL HEAP 01752
JOHN HEATH 04103
JUDY HERRON 91789
S. J. HIGGINS CA9 3LP UNITED KINGDOM
TERUGO HIKITA 158 JAPAN
BUZZ HILL 98632
BRUCE HILLEBRAND 01754
ED HIRAHARA 83704
TIHON HOARD 55414
C. A. R. HOARE OX2 6PE UNITED KINGDOM
FRED H. HOFFMAN 19117
WILLIAM HOLMES 63110
JAMES W. HOLT JR. 40222
JILL A. HOLTZ 63166
WILLIAM C. HOPKINS 19335
MARK HORTON 53706
DAVID A. HOUGH 24502
MICHAEL A. HOUGHTALING 85712
D. J. HONORTH UNITED KINGDOM
PEI HSIA 35801
PETER TAN-TEK HSU 55455
HARTMUT G. HUBER 22448
GARY HUCKABAUGH 73501
JACK HUGHES K7L 3H6 CANADA
PHIL HUGHES 98507
EDWARD G. HUMPHREY 78769
H. A. HUSBAND UNITED KINGDOM
DANIEL G. HYNE 17837
ELIZABETH IARRA 91360
MICHAL ICHLEWSKI 00590 POLAND
BERT INCARFIELD 77459
DAVID I. INTERSTONE 90250
T. M. V. IRISH NP6 758 UNITED KINGDOM
PORTIA ISAACSON 75080
HANNU JAAKKOLA SF-33570 FINLAND
TIMOTHY H. JACKSON 94306
STEVE JAY 85721
ROY JEFFRIES 93917
JOSEF JINICH 7 CZECHOSLOVAKIA
DOUGLAS N. JOHNSON 94599
LYTLE JOHNSON 80033
PAUL D. JOHNSON 75006
RALPH JOHNSON 61401
ROBERT T. JOHNSON 87545
S. JOHNSON 85002
JOSEPH B. JOLDA IRELAND
ALAN JONES LSN 1K7 CANADA
RUSSELL JONES M5M 3W6 CANADA
MARK JUNGWIRTH 93010
MIKE KAYRAD 55413
KENNETH KAPLAN 50304
HEIKKI KASKELMA SF-00130 FINLAND
RICHARD KAUFMAN 92093
FRED J. KELLER 99210
MIKE KERSEY 77098
ADHMAN KHAN PAKISTAN
JOHN H. KILFOIL 95126
ERICH R. KNOBIL 33313
TOM KNOEHE 92109
STEPHEN KOCH 01730
JAMES R. KOCHANOWICZ 60601
JUDITH KOVETZ ISRAEL
JEFF KRAVITZ 11771
DIETRICH KREKEL D-5900 GERMANY
RICHARD M. KRUEZER
CHARLES KUHLMAN 10007
DARRYL KUHNS 89503
KHAI-SAND LAH D-4440 GERMANY
K. LANG B15 2TT UNITED KINGDOM
GUY LAPALME H3C 3J7 CANADA
G. P. LAPPIN UNITED KINGDOM
JOHN LATRASH 20910
CHARLES L. LAWSON 91103
HENRI A. LE FRIANT 70118
ROHARD LEBEL 02139
RICHARD LEBLANC 30332
ROLAND L. LEE 94121
R. GARY LEE 32106
WILLIAM J. LEE 55423
P. J. LEGGE M60 1QD UNITED KINGDOM
CLARENCE LEFMAN 55364
CHARLES T. LEIS 94087
WILL LELAND 53715
JOE LEMNER 38163
ALAN M. LEGGOLD 13250
KEN LESSEY 97051
JAMES D. LEZT 92805
DAVID J. LEWIS 14830
HORNG JHN LIANG 20854
PING K. LIAO 94545
JACK LIEBSCHUTZ 02139
JOHN E. LIND 55455
FRANK LINDSAY 22209
CHUI FAN LIU 60005
WILLIAM R. LLOYD 94115
CARLO LOVICERO L5L 2E9 CANADA
STEPHEN A. LONGO 19141
ANTHONY P. LUETTO 77843
CLARENCE W. LYBARGER 55901
STUART LYNNE CANADA
BRUCE MAC ANASPJE 21914
D. W. MACLEAN S7H 0V0 CANADA
ORLANDO S. MADRIGAL 95929
KEVIN T. MAHOEY 01742
DENNIS J. MAINE 92717
STEPHEN MANN 01730
DUANE F. MARBLE 14269
THOMAS A. MARCIANAK 20014
B. L. MARKER 35805
BILL MARSHALL 03068
WILLIAM J. MARSHALL 98033
S. MATTHEWS HAJ 1N1 CANADA
STEVE MC FERRIN 94086
FRANK E. MCGARRRY 20771
JAMES PATRICK MCGEE 77025
JAMES A. MCGILLICHRAY 19403
DAVID J. MC KEZE 22209
IRVINE L. MCKNIGHT 94040
WILLIAM L. MCLAIN III 27450
RONALD P. MCGRANEY 70360
JOHN MEDCALF 94707
MICHAEL MEEHAN 02138
JIM MERRITT 94086
HANS J. METZENDORF CH-6900 SWITZERLAND
KURT METZGER 48105
JOSEPH MAYER 17557
R. M. MILLER 75229
PATRICIA R. MOHLNER 80523
ROLF MOLLICH DK-2730 DENMARK
UFFE NOLLER DK-9220 DENMARK
ALLAN MOLUP 48864
CECIL A. MOORE 95051
FREDMAN L. MOORE 48859
RAYMOND MOREL CH-1224 SWITZERLAND
CLEMENS MORITZ 01720
L'WALL MORRELL 94114
GREG MORRIS 54701
JWO-WU'YOU CHINA
GLEN R. J. MUILES 10804
ANN MURPHY 20742
CHARLES MYERS 60611
DONALD V. MYHRA 92705
J. P. M. STOFBERG 19422
GERALD NADLER 01351
OLAV NAESS NORWAY
JOHN NAGLE 95051
GEORGE NACE 85898
T. A. NARTKER 87801
ED NAYLOR 78761
PETER A. NAYLOR 19422
HEIDI L. NEUBAUER 61901
HARTIN NICHOLS 07301
DANIEL NICHOLSON 55901
DENNIS NICHOLSON 19401
HIRAKI NISHIOKA 544 JAPAN
TOMY NOE 91711
JOHN NOLD 15701
HELVIE L. NORRELL 90070
BARBARA K. NORTH 66502
LARRY T. NOVAK 75235
WILLIAM I. NOVICKI 60201
M. MUN SWLP ART UNITED KINGDOM
FRANK NUSSBAUM 69626
FRANK W. OCHSLSLI 94611
DAVID P. OHL 95914
ERIC OLSON 92713
GENE H. OLSON 55419
JAMES B. ORNEY 58022
P. E. OSWON HJ3 7ST UNITED KINGDOM
SUSAN S. OVIICKI 94305
JAMES N. O'BRIEN 97229
STEVE O'KEEFE 20229
JACK PAGE SINGAPORE
W. O. PAINE 91103
R. L. PALASKE 44539
RICHARD PALZHIK 95014
PAUL J. PANTANO 95051
T. L. (FRANK) PAPPAS 19083
M. L. PATRICK R20 1LL UNITED KINGDOM
DAVID H. PECK 01535
JIM PECK 16142
ROBERT C. PERLE 08753
BERNARD PERRITTE F-92803 FRANCE
RONALD H. PERROTT 94086
DAVID PESCE 44119
JAMES L. PETERSON 78712
SUSAN A. PETERSON 55432
CHRIS K. PHILLIPS 94305
DAVID H. PHILLIPS 78736
DAVID PICKENS 80302
THOMAS PINNOT 19530
FRED POSPESCHIL 68005
DAVID POWERS 2049 AUSTRALIA
KARL PRAGERSTORFER A-4080 AUSTRIA
GERH. H. PRIESTMAN 93129
FRANK PROSSER 47401
ANDREW S. PUCHRIK 47119
J. R. PUGH S1 1WB UNITED KINGDOM
SHING-KIT PUN 95926
ABBAS RAFLI 73019
ROBERT J. RAKPR 94104
STEVEN R. RAKITEN 06095
TIM RARD 06258
ROBERT RAMSDELL 43202
DONALD D. REDDING 48175
WERNER REHMELE D-8000 GERMANY
JOHN H. REHMER 48197
TIMOTHY P. ROBERTS 10509
KEN ROBINSON 2033 AUSTRALIA
D. J. ROBSON UNITED KINGDOM
MICHAEL ROBBY 50701
JAMES D. ROGAN 48136
J. S. ROHL 60009 AUSTRALIA
JOY D. ROLAND 78205
GENE ROLLINS 11794
J. ROSCOT UNITED KINGDOM
PETER H. ROTH 22030
RICHARD ROTH 06877
R. WALDO ROTH 46989
H. J. ROWE LEI 7RH UNITED KINGDOM
HERB RUBENSTEIN 80401
BEARDSLEY RUPIL 2ND 20008
HOWARD RUTLEGER 60023
JOHN L. RUTIS 97106
DAVID W. SALLUNH 93454
ANN D. SANDERSON 32884
TOM SANDERSON 91311
HAROLD S. SCHECHTER 11418
ALAN M. SCHLENGER 95064
CH. SCHLIER D-7800 GERMANY
RICHARD SCHLOTTFELDT 55435
ANA-MARIA SCHMIT CH-1007 SWITZERLAND
ED SCHOFELL 95051
ROBERT L. SCHOENFELD 10921
PETER L. SCHULTZ CH-8006 SWITZERLAND
JIM SCHULTZ 95014
WILLIAM L. SEIFERT 87545
MICHAEL SETTLE 75229
LEONARD SHAPIRO 58102
T. K. SHARPLESS 10021
TIMOTHY SHAW 20016
ALAN M. SHERKOW 53212
BRUCE SHERRY 95050
R. G. SHERRY 98107
THOMAS E. SHIELDS 77005
KEITH ALLAN SHILLINGTON 92093
CHARLES B. SHIPMAN JR. 22027
KIM L. SHIVELEY 75231
ARNOLD SHORE 22032
MICHAEL L. SIMON 94618
LESLIE L. SIFTER 02181
JOHN SIGLE 78284
GEORGE A. R. SILVER 47374
GREN SIMMONS 22030
TIMOTHY C. SIMMONS 72143
SIMON 2308 AUSTRALIA
ANDREW SINGER 02546
ROGER SIPP 92460
DAVID SKINNER 97330
LEO J. SLECHTA 55165
JOSEPH W. SMITH 92127
TOM SNYDER 92634
DAVID SOLOMOHIT 02155
RICHARD P. SPRACHE 92714
ROB SPRAY 75207
R. P. STARK 15021
JAMES A. STARK 94609
R. A. STASLOR 13098
MICHAEL K. STAUFFER 95051
GREG STEPHEN 62026
WIL STEVENS B-2510 BELGIUM
JIM STEWART 08854
ZHAHAI STEWART 80306
RICHARD A. STONE 55435
QUENTIN F. STOUT 13901
H. J. STRATFORD-COLLINS 06720
CORDON STUART W8P 5J2 CANADA
INDRO S. SUWANDI INDONESIA
MARK A. SWANSON 02174
ROBERT M. SWENBY 38103
E. SWEET 44199
KEN SYLVESTER YIA 3P5 CANADA
DAVID TARANAR 01581
H. TAYLOR KIN 6M5 CANADA
R. F. TAYLOR 90026
DON TERMILLIGER 97005
DANIEL THALMAN H3C 3J7 CANADA
K. TIZZARD EK4 4PU UNITED KINGDOM
HOWARD E. THOMPSON 15701
ROY TOUFEAU 59812
MIKE TRAVIS 95051
TOM A. TROTTER HAR 1V2 CANADA
JAMES TRURNLOO 19713
JIM TSEVODS 15213
HOWARD L. TURETZKY 80220
GERALD P. UHLIG 55112
S. H. VAIDYA 411007 INDIA
TOM VAN DER HOEVEN THE NETHERLANDS
STANLEY C. VESTAL 55413
JOHN VISEL 23061
LES VOGEL 93940
PATRIK WAHREN S-102 62 SWEDEN
SCOTT WAKEFIELD 94305
JOHN WALKER 94941
JUSTIN C. WALKER 20234
TIM WALSH 11580
P. R. WALTON KT3 3E2 UNITED KINGDOM
C. DUDLEY WALKER 95030
SANDY WARSILAW 12301
MARK S. WATERBURY 22044
ANNA WATSON 32407
CHTP WEKMS 97331
STEPHEN J. WEINBERGER 64138
STEVEN W. WEINGART 01581
LAUREN WEINSTEIN 90025
MICHAEL D. WEINSTOCK 32548
DAVID H. WEICH 92501
JAMES H. WELLS 92634
D. R. WESTLUND K9K 1J1 CANADA
HARVIN WHITE 97077
RONALD C. WHITES 92701
HAREK WIECHULA B2C 1C0 CANADA
JACK M. WIJERNA 60532
J. F. WILKES THE NETHERLANDS
ROY A. WILSKER 02114
MICHAEL WIMBLE 52404
BILL WINSPUR R3E 0W3 CANADA
JOHN WITHROW 01752
ERIC WOODSBERG 94618
BRUCE WOLFE 94965
SHARLEEN WONG 94134
STEPHEN C. WOOD 87108
ARDEN WOOTTON 64151
FULTON URIGHT JR. 86301
BRUCE YALE 92506
M. J. L. YATES GL52 5AJ UNITED KINGDOM
JAMES CRAIG ZIEGLER 38138
DAVID J. ZOOK 60626

- 12244
23601
24502
27410
27514
29208
30332
30332
32306
32407
32548
32601
32901
33113
35223
35801
35805
37916
38103
38138
38163
38677
40222
43923
43923
43923
43923
45241
46989
47119
47374
47401
47401
47904
48012
48103
48105
48106
48109
48197
48859
48864
50304
50701
52242
52404
52543
53212
53706
53706
53715
54701
55112
55113
55113
55119
55165
55243
55364
55408
55413
55414
55419
55423
55432
55433
55440
55455
55455
55455
55455
55455
55901
55901
55901
58182
59812
60025
60201
60439
60532
60601
60611
60626
60919
61401
61801
61801
62026
62704
63110
63166
64138
64151
66502
68762
68005
68134
68154
68508
68583
68588
69626
70118
70360
72143
73209
73202
73501
74104
74171
75006
75207
75229
75229
75231
75231
75238
76010
77005
77025
77098
77459
77840
77843
78205
78284
- ATTN: DEPT. OF COMPUTING ACTIVITIES/ VIRGINIA COLLEGE
RICKY W. BUTLER/ 132 WOODS ROAD/ NEWPORT NEWS VA 23601/ (804)
DAVID A. HUGHES/ 223 BRYANT RD./ LYNNBURG VA 24502/ (804) 237-
WILLIAM L. HORN/ 211 CONN SWY/ GUILFORD COLLEGE/ 5000 W. FR
STEVEN M. LUDWIG/ DEPT. OF COMP. SCI./ U OF NORTH CAROLINA
ROBERT L. CARROLL/ DEPT. OF MATH. AND COMP. SCIENCE/ UNIV. OF S
ATTENTION: JERRY W. SEGERS/ OFFICE OF COMPUTING SERVICES/ GEOR
RICHARD LEBLANC/ SCHOOL OF INFO. AND COMP. SCI./ GEORGIA TECH
G. GARY LEE/ DEPT. OF MATH./ 113 LOVE BUILDING/ FLORIDA STATE
ANNA WATSON/ 3705 DELWOOD DRIVE/ PANAMA CITY FL 32407/ (904) 2
MICHAEL D. WEINSTECK/ 112 KNOXWOOD WAY/ FT. WALTER FL 32548
SCOTT CRUMPTON/ 11130 N.W. 9 PLACE/ GAINESVILLE FL 32601/ (904)
T. M. CAMRON/ TR/1/ HARRIS ESD/ P.O. BOX 37/ MELBOURNE FL 32
ERIC R. KNOWL/ PRODUCT DEVELOPMENT/ SYSTEMS ENGINEERING LABS
JEFFREY W. GRAHAM/ GRAHAM COMPUTER ENTERPRISES INC./ 3 OFFICE
PEI HSIA/ 3300 O'HARA DR./ HUNTSVILLE AL 35801/ (205) 895-6088
L. B. MARKER/ DEFENSE AND SPACE SYSTEMS/ TR/1/ 7702 GOVERNORS D
ATTENTION: GORDON R. SHERMAN/ COMPUTER CENTER/ 200 STOKELY GYM
ROBERT M. SWEENEY/ ELLERS FARMING OAKLEY CHESTER & RIKE/ 722 F
JAMES CRAIG ZIEGLER/ 1455 LANCASTER DR./ MEMPHIS TN 38138
JOE LENKEL/ DEPT. OF PATHOLOGY/ UNIV. OF TENNESSEE/ 858 MADISON
ATTN: DEPT. OF COMPUTER SCIENCE/ U OF MISSISSIPPI/ UNIVERSITY
JAMES W. HOLT JR./ 7705 NORWOOD DR./ LOUISVILLE KY 40222/ (502)
JIM CAMERON/ DEPT. OF MATH SCIENCE/ DENISON UNIVERSITY/ GRANVI
ROBERT KANSDEL/ 619 HARLEY DRIVE APT. 5/ COLUMBUS OH 43202
RICHARD E. FARMER/ 239 GUMTAM ROAD/ COLUMBUS OH 43214/ (614) 2
DAVID PESIC/ 2130 MILLER AVENUE/ EUCLID OH 44119/ (216) 486-8
E. L. PALASER/ DEPT. OF APPLIED SCIENCES/ BOULDER CREEK STATE
ATTN: BETTE HOLLING-LIBERMAN/ TECHNICAL INFORMATION CTR-ELECT
FRANCIS M. BEARDEN/ DATA SYSTEMS/ CINCINNATI ELECTRONICS CORP.
R. WALDO ROTH/ COMPUTER SCIENCE DEPT./ TAYLOR UNIVERSITY/ UPLAN
ANDREW S. PUCHRIK/ 1111 ANDREA DR./ FLOYDS KNOBS IN 47119/ (50)
GEORGE A. R. SILVER/ DEPT. OF HISTORY/ EARLHAM COLLEGE/ RICHMO
ALN BARDIN/ KRUBEL COMPUTING CENTER/ HPER BUILDING/ INDIANA UN
FRANK PROSSER/ COMPUTER SCIENCE DEPT./ 101 LINDLEY HALL/ INDI
HOWARD CUNNINGHAM/ 914 NORTH 21ST STREET/ LAFAYETTE IN 47904/
STEIN DEVILIN/ 6854 CEDARBROOK/ BIRMINGHAM MI 48210
KURT METZGER/ 48 CLOVERDALE/ ANN ARBOR MI 48105
DONALD D. REDDING/ HDSI/ 4251 PLYMOUTH ROAD/ ANN ARBOR MI 4810
JAMES D. NOGAN/ CONSHARE INC./ P.O. BOX 1588/ ANN ARBOR MI 481
E. SWEET/ COMPUTING CENTER/ UNIV. OF MICHIGAN-NORTH CAMPUS/ 10
JOHN H. REMERS/ DEPT. OF MATHEMATICS/ EASTERN MICHIGAN UNIV./
FREDMAN L. MOORE/ DEPT. OF COMPUTER SCIENCE/ 203-S PIERCE HALL/
ALLAN MOLUF/ 2317 KNOB HILL APT. 9/ OKEMOS MI 48864
KENNETH KAPLAN/ MICROWARE SYSTEMS CORP./ P.O. BOX 954/ DES MOI
MICHAEL ROBBY/ 322 DEVONSHIRE/ WATERLOO IA 50701/ (319) 233-57
ATTN: SERIALS DEPT./ UNIVERSITY LIBRARIES/ UNIVERSITY OF IOWA/
MICHAEL WIMBEL/ 6026 UNDERWOOD AVE. S.W./ CEDAR RAPIDS IA 5240
DAVID E. ANDERSON/ JOHNSON CONTROLS INC./ 507 E. MICHIGAN AVE.
ALAN W. SHERKON/ 1211 EAST SINGER CIR. APT. #4/ MILWAUKEE WI 5
ATTN: FRIDA S. COHN/ ACADEMIC COMPUTING CENTER/ U OF WISCONSINI
MARK HORTON/ COMPUTER SCIENCE DEPT./ UNIV. OF WISCONSIN/ 1210
WILL LELAND/ 445 N. LAKE ST./ MADISON WI 53715/ (608) 257-4035
GREG MORRIS/ 1705 WILSON ST./ EAU CLAIRE WI 54701/ (715) 835-6
GERALD P. UHLIG/ 3545 OWASSO ST. APT. 110/ SHOREVIEW MN 55112/
ATTN: INFORMATION SERVICE CENTER/ SPERRY-UNIVAC/ 2276 HIGH CREE
DAVID GARDNER/ 1730 LARSENTEUR - APT. 3C/ ST. PAUL MN 55113/ (612)
GARY BECKWITH/ 2199 GLENVIEW AVE./ ST. PAUL MN 55119/ (612) 8
LEO J. SLEIGHT/ 477 WATSON AVE. BOX 325/ ST. CLOUD MN 56301/
CLIFFORD GERSTENBERG/ DEFENSE SYSTEMS/ M11-2120/ HONEYWELL/ 600
CLARENCE LEDIAN/ 5926 GUMWOOD ROAD/ MOUND MN 55364/ (612) 472-1
ABUHL RASAQ BELLO/ P.O. BOX 8681/ MINNEAPOLIS MN 55408/ (612) 330-4
MIKE KAMMAY/ HONEYWELL AVIONICS/ 2600 RIDGWAY PKWY/ MINNEAPOLIS
STANLEY C. VESTAL/ MS 2340/ HONEYWELL INC./ 2600 RIDGWAY PKWY.
THOM HOARD/ P.O. BOX 14413/ MINNEAPOLIS MN 55414/ (612) 376-62
GENE H. OLSON/ 5149 ALDRICH AVE. S./ MINNEAPOLIS MN 55419/ (612)
WILLIAM J. LEE/ 7320 FIKST AVE. SO./ RICHFIELD MN 55423/ (612)
SUSAN A. PETERSON/ SOFTWARE DEVELOPMENT/ M.S. E-360/ DATA 100
RICHARD SCHLOTZLEDT/ 7330 GALLAGHER DR. - APT. 255/ EDINA MN 5
RICHARD D. 100 CORNWELL/ 775 WASHINGTON AVE. SO./ MINNEAPOLIS M
CHRISTOPHER AHLEY/ SSRFC/ 25 BLEDEN HALL/ UNIV. OF MINNESOTA/
ATTN: SPECIAL INTERACTIVE COMPUTER LAB/ 143 SPACE ENGINEER UNIV.
SCOTT S. BERTILSON/ UNIVERSITY COMPUTER CENTER/ 227 EXP. ENGR.
PETER YAN-TEK HSI/ COMP. SCI. DEPT./ 114 LIND HALL/ U OF MINNE
JOHN E. LIND/ 363 TERRITORIAL HALL/ UNIVERSITY OF MINNESOTA/ E
DENNIS S. DELORNE/ 1407 13TH AVENUE NE/ ROCHESTER MN 55901/ (507)
CLARENCE W. LYBARGER/ 3513 VILLA ROAD N.W./ ROCHESTER MN 55901/
DANIEL NICHOLSON/ 2112 17TH AVE NW/ ROCHESTER MN 55901/ (507) 2
LEONARD SHARP/ DEPARTMENT OF NORTH DAKOTA STATE UNIV./ PAR
BOU TOUZEAU/ COMPUTER SCIENCE DEPT./ UNIV. OF MONTANA/ MISSOUL
CHUI FAN LIU/ 1603 S. HIGHLAND AVE./ ARLINGTON HTS IL 60005
HOWARD RUTZEN/ 1101 ARNOR/ GLENVIEW IL 60025/ (312) 234-3400 (WORK)
WILLIAM I. NOWICKI/ CSRL TECH/ 8626/ NORTHWESTERN UNIV./ 2145
RICHARD D. GEORGE/ ARGONNE NATIONAL LABORATORY/ 9700 S. CAS
JACK M. WIERDA/ DEPT. 7346/ WESTERN ELECTRIC COMPANY/ 4513 WE
JAMES R. KOCHANOWICZ/ DEDICATED SYSTEMS INC./ 180 N. MICHIGAN
CHARLES MYERS/ DATALOGICS/ 325 W. HURON/ CHICAGO IL 60611/ (31
DAVID J. ZOOK/ 1100 W. PRATT/ CHICAGO IL 60626
MICHAEL FAY/ STELLE STATION/ CAREY IL 60919
RALPH JOHNSON/ 1502 N. BRADY/ CHICAGO IL 61401/ (309) 342-30 98
DONALD D. BURN/ 410 WEST ILLINOIS/ URBANA IL 61801/ (217) 367-5362
REIDI L. NEUBAUER/ COORDINATED SCIENCES LAB/ UNIV. OF ILLINOIS -
GREG STEPHEN/ DEPT. OF MATH. STUDIES/ S. ILLINOIS UNIV. AT EDW
MIKE HARRIS/ APT. 4/ 309 WEST EDWARDS/ SPRINGFIELD IL 62704/ (217)
WILLIAM HOLMES/ DEPT. OF BIOCHEMISTRY/ WASHINGTON UNIV. MED. S
JILL A. HOLTZ/ PROGRAMMING SCIENCES DEPT./ BLDG. 105 LEVEL 4/E
STEPHEN J. WEINBERGER/ 8340 B. HILLCREST RD./ KANSAS CITY MO 6
ARDEN WOOTTON/ 5904 N.W. LENOX AVE./ KANSAS CITY MO 64151/ (81
BARBARA K. NORTH/ 1516 NICHOLS ST./ MANHATTAN KS 66502/ (913) 5
FELIX BRENER/ COMPUTER SCIENCE/ PITTSBURGH STATE UNIVERSITY/ P
FRED POSPESCHIL/ 3108 JACKSON ST./ BELLEVUE NE 68005/ (402) 29
EARL N. CHRISTIANSEN/ BASCO COMPANY/ 9685 AMES/ OHAMA NE 68134
CHARLES J. GIBBONS/ SUITE 8/ APPLIED COMMUNICATIONS INC./ 1084
K. S. BHASKAR/ ACADEMIC COMPUTER SERVICES/ 225 WYBRASKA HALL/ U
D. F. COSTELLO/ ACADEMIC COMPUTING SERVICES/ UNIVERSITY OF NEB
GEORGE NAGY/ DEPT. OF COMP. SCI./ 110 FERGUSON HALL/ U OF NEBR
FRANK NUSSBAUM/ DEPT. OF MATH. SCIENCES/ IOWA UNIV./ 6525 N. S
HEMEL A. LE FRIANT/ 8200 APRICOT ST./ NEW ORLEANS LA 70118/ (504)
RONALD P. MCRAVENY/ STATION 1/ P.O. BOX 10097/ HOUA LA 70360/ (504)
TIMOTHY W. SIMMONS/ HARDING COLLEGE/ BOX 1193 STATION A/ SEAR
ARNAZ RAHIL/ P.O. BOX 218/ DEPT. UNIV. OF OKLAHOMA/ 202 N. BOYD/ N
REPOND BOND/ ESSAY CORP./ 100 PARK AVE./ OKLAHOMA CITY OK 7310 2
CARY HUCKABAY/ DEPT. OF MATHEMATICS/ CAMERON UNIV./ LAUFON OK
C. BAILEY/ BAILEY AND ASSOCIATES/ 1144 S. ATLANTA/ TULSA OK 74
ANTHONY BJEASTEDT/ ORAL ROBERTS UNIVERSITY/ BOX 1127/ TULSA OK
PAUL D. JOHNSON/ 2009 WEDGEWOOD/ CARROLLTON TX 75006/ (214) 24
PORTIA ISAACSON/ THE MICRO STORE/ 634 S. CENTRAL EXPY./ RICHAR
ROB SPRAY/ MS 420-160/ ROCKWELL INTERNATIONAL/ P.O. BOX 10462/
E. M. MILLER/ INTERNATIONAL COMPUTER PRODUCTS/ 2925 MERRRELL R
MICHAEL SETTLE/ INTERNATIONAL COMPUTER PRODUCTS/ 2925 MERRRELL
KIM L. SHIVELET/ 7777 HANDEVILLE LANE APT. 221/ DALLAS TX 752 31
LARRY T. HOWAR/ MS 348/ TEXAS INSTRUMENTS/ P.O. BOX 35486/ DAL
JOE COLENTINE/ 7709 QUEENS GARDEN DR./ DALLAS TX 75248 (214)
BEN CLARK/ P.O. BOX 517/ 461 SAUFERON TX 76011/ (817) 481-5429
THOMAS E. SHIELDS/ SOFTWARE RESOURCES/ P.O. BOX 25210/ HOUSTON
JAMES PATRICK MCGEE/ P.O. BOX 20223/ HOUSTON TX 77025/ (713) 6
MIKE KERSEY/ 1850 W. MAIN #1/ HOUSTON TX 77098/ (713) 526-8658
BERT INCAPLETT/ ACTION COMPUTER SERVICES/ 2202 MUSTANG SPRING
JERRY R. BROOKS/ 1201 HIGHWAY 30 - APT. 153/ COLLEGE STA.
ANTHONY P. LUCIDO/ COMPUTING SCIENCES DEPT./ TEXAS A&M UNIV./
JOE B. HOLAHD/ MICRO MART/ 1015 NAVARRO/ SAN ANTONIO TX 78205/
JOHN STIGLE/ TRINITY UNIV./ P.O. BOX 237/ SAN ANTONIO TX 78284/

UNITED KINGDOM D. J. ROSS/ DEPT. OF COMPUTER STUDIES/ UNIV. OF BURL/ COTTIN CHAM ROAD / S. WIMBORNE/ HULL ENGLAND/ UNITED KINGDOM
 UNITED KINGDOM ANN V. BARRON/ 19 DE VITRE STREET/ LANCASTER LANCS/ UNITED KINGDOM
 UNITED KINGDOM J. ROSCOR/ COMPUTER DEV. DIV. - SYSTEMS ENG./ I. C. L./ WENLOCK WAY / WESTGORTON/ MANCHESTER ENGLAND/ UNITED KINGDOM/ 061 223 1394 X2589
 UNITED KINGDOM C. F. LAPPIN/ MATH. AND COMPUTER STUDIES/ PORTADOWN TECH. COLL. GLE/ LURGAN ROAD / CO. ANTRIM/ PORTADOWN N. IRELAND/ UNITED KINGDOM
 UNITED KINGDOM M. A. HUSSEIN/ AUTOTEST/ HANCOCK INSTRUMENTS/ LONGACRES/ ST. A. LEDES HERTS/ UNITED KINGDOM
 BA2 7AY UNITED KINGDOM P. CARR/ SCHOOL OF MATHEMATICS/ UNIVERSITY OF BATH/ BATH ENGLA RD BA2 7AY/ UNITED KINGDOM
 BT7 1RN UNITED KINGDOM JOHN SELWY/ DEPT. OF COMPUTER SCIENCE/ THE QUEEN'S UNIVERSITY/ BELFAST N. IRELAND BT7 1RN/ UNITED KINGDOM
 B15 2TT UNITED KINGDOM K. LANG/ THE COMPUTER CENTRE/ UNIV. OF BIRMINGHAM/ ELMS ROAD/ BIRMINGHAM ENGLAND B15 2TT/ UNITED KINGDOM/ 021-472-1301 X2233
 B20 1LL UNITED KINGDOM H. L. PATRICK/ 42 CHATSWORTH AVE. / BANBURNHURST WOOD/ BIRMINGHAM 4 ENGLAND B20 1LL/ UNITED KINGDOM
 CA9 3LP UNITED KINGDOM S. J. HIGGINS/ WORLANDS SOFTWARE LTD./ WENTHEAD/ ALSTON CUMBR IA CA9 3LP/ UNITED KINGDOM
 DD2 3XJ UNITED KINGDOM R. B. T. SLACK/ ENGINEERING DIV./ NOR LTD./ KINGSHAY WEST/ DUN DEE SCOTLAND DD2 3XJ/ UNITED KINGDOM
 H24 APU UNITED KINGDOM K. TIZZARD/ DEPT. OF M.S.O.R./ UNIV. OF EXETER/ STREATHAM COURT/ EXETER ENGLAND H24 APU/ UNITED KINGDOM
 FK9 4LA UNITED KINGDOM ROBERT G. CLARK/ COMPUTING SCIENCE DEPT./ UNIVERSITY OF STIRLING/ STIRLING SCOTLAND FK9 4LA/ UNITED KINGDOM/ STIRLING 3171 42570
 LS2 5AJ UNITED KINGDOM M. J. L. TATES/ 7/0603 266HQ/ GCHQ OAKLEY/ CHELSEA ROAD/ ENGLAND GL52 5AJ/ UNITED KINGDOM
 T23 3EZ UNITED KINGDOM P. R. WALBY/ 57 BARN MEADOW LANE / GREAT BOOKHAM/ SURREY ENGLAND KT23 3EZ/ UNITED KINGDOM
 LE1 7RH UNITED KINGDOM ATTN: DEPT OF BIOCHEMISTRY/ ADRIAN BUILDING/ UNIVERSITY OF LEICESTER/ LEICESTER ENGLAND LE1 7RH/ UNITED KINGDOM
 LE1 7RH UNITED KINGDOM H. J. ROWE/ COMPUTER LABORATORY/ LEICESTER UNIVERSITY/ LEICESTER ENGLAND LE1 7RH/ UNITED KINGDOM/ LEIC. 50000
 L69 3JK UNITED KINGDOM H. D. BEIN/ COMPUTER LABORATORY/ UNIVERSITY OF LIVERPOOL/ P.O. BOX 147/ LIVERPOOL ENGLAND L69 3JK/ UNITED KINGDOM
 M13 9PL UNITED KINGDOM TONY ARDITANI/ DEPARTMENT OF COMPUTER SCIENCE/ THE UNIVERSITY/ OXFORD ROAD/ MANCHESTER ENGLAND M13 9PL/ UNITED KINGDOM/ 761-273 7121 X5546
 M20 1QD UNITED KINGDOM RICHARD CLAYTON/ 20 BALDOCK ROAD/ MANCHESTER ENGLAND M20 1QD/ UNITED KINGDOM
 M60 1QD UNITED KINGDOM E. J. LEGGE/ DEPT. OF PHYSICS/ U.M.I.S.T.-/ SACKVILLE STREET/ MANCHESTER ENGLAND M60 1QD/ UNITED KINGDOM
 M99 1LH UNITED KINGDOM ATTN: THE LIBRARIAN/ ENGINEERING RESEARCH STATION/ BRITISH GAS CORPORATION/ KILLINGWORTH/ NEWCASTLE/TWE ENGLAND M99 1LH/ UNITED KINGDOM
 NP6 7BB UNITED KINGDOM T. M. N. IRISH/ 5 HORSE MAY SEDBURY/ CHEPSTOW OUBT NP6 7BB/ UNITED KINGDOM
 NP3 7ST UNITED KINGDOM P. E. OSNOD/ COMPUTER SCIENCE DEPT./ WESTFIELD COLLEGE/ KIDDER POLE AVENUE/ LONDON ENGLAND NP3 7ST/ UNITED KINGDOM/ 01-435-7141 X521
 OX2 6PE UNITED KINGDOM C. A. R. ROARE/ PROGRAMMING RESEARCH GROUP/ OXFORD UNIVERSITY COMPUTING LABORATORY/ 45 BANBURY ROAD/ OXFORD ENGLAND OX2 6PE/ UNITED KINGDOM
 SE1 0AA UNITED KINGDOM R. COLLE/ DEPT. OF MATH. SCIENCES AND COMPUTING/ POLYTECHNIC OF THE SOUTH BANK/ BOROUGH ROAD/ LONDON ENGLAND SE1 0AA/ UNITED KINGDOM
 SW1P 4RT UNITED KINGDOM M. NUNN/ 407/ CCA/ RIVERSWALK HOUSE/ LONDON ENGLAND SW1P 4RT/ UNITED KINGDOM
 S1 1UB UNITED KINGDOM J. R. PUGH/ DEPT. OF COMPUTER STUDIES/ SHEFFIELD CITY POLYTECHNIC/ HERIOT HOUSE - FORD HILL/ SHEFFIELD ENGLAND S1 1UB/ UNITED KINGDOM
 WC1A 0AH UNITED KINGDOM ROBERT O. HARRIS/ COMPUTER CENTRE/ UNIVERSITY COLLEGE - LONDON/ 19 GORDON ST./ LONDON ENGLAND WC1A 0AH/ UNITED KINGDOM/ 01-387 0858



APPLICATIONS

PLEASE SUBMIT ALL PROGRAMS, SOFTWARE TOOLS, ALGORITHMS, etc. for this section to Rich. Thanks, Andy.

Editor: Rich Cichelli 901 Whittier Drive Allentown, PA 18103 USA (phone: 1-215-253-6155 work 1-215-797-9690 home)

We decided to create a new section for printing Pascal source programs for various applications including software tools and algorithms. Additionally here, we will print news of significant applications programs written in Pascal.

Jim Miner suggested we should index each program so that they may be easily referenced for corrections and criticisms.

Arthur Sale is very enthusiastic about the algorithms section. He suggested that we allow for 1) the provision for certification of the program by unrelated persons, with clear identification of system used; and

- 2) critiques of the program for a) standards conformance, or b) style, or c) algorithm, or d) output convenience and general design.

We'll number programs starting with P-1, software tools starting with S-1, and algorithms starting with A-1.

New Software Section

I am looking forward to your new software section in News, partly because I may see something really good there, but even more for the educative role I think it will play among Pascal users. I think back to the large influence that the CACM Algorithms section had on Algol stylistics and their propagation, and on the appreciation of subtle points of the FORTRAN standard.

So besides the big program things (which I shall enjoy perusing for their non-portable features), may I suggest that News could serve as a useful vehicle for procedures and algorithms of very common use? For example, what about a portable lexical analyzer for reference PASCAL (as in the Report) which itself had no assumptions about wordsize or setsize beyond those likely to hold in all compilers? Or a tree-balancer? And so on. I'm therefore enclosing a little thing to stir up those other contributors you have: a procedure to produce a 'Pascal standard time and date log record'. You will see that a little machine-dependency is not harmful to communication, provided it is collected and clearly identified as such.

Arthur Sale

Arthur Sale.

** NEWS **

In PUGH # 6, John Banning wrote to solicit programs for his empirical study of Pascal programs. On 78/01/14 he wrote that he had a change in thesis topic which delayed his performing the study. He enclosed a listing of a 12000-line Pascal program which analyzes Pascal programs for style and behavior.

In PUGH # 11, Prof. David Barron announced the formation of a numerical library project. He has recently been in touch with Jan Kok and Reind P. van de Riet of Vrije Universiteit and Mathematisch Centrum in Amsterdam, The Netherlands, who have constructed a large library for Pascal on CDC-6000 series machines.

When numerical analyst Gilbert W. Stewart heard about the project, he was interested, and gave the advice that algorithms beware of the matrix storage differences between Pascal and FORTRAN (row-wise versus column-wise).

Applications

Applications

ALGORITHMS

A - 1 Random Number Generator

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

University of Lancaster

Head of Department: J. A. Llewellyn B.Sc., M.Phil., F.B.C.S., F.I.M.A.

30th November 1977.

Dear Andy,

In case anyone is interested, as I am, in using Pascal for simulation purposes, I present a Pascal random number generator based on the feedback shift register pseudorandom number generator algorithm given by Whittlesey [1], for a 16-bit word size machine.

Note that:

- i) for any other word size, the constants 'pshift', 'qshift' and 'big' must be changed; (see Lewis [2])
- ii) bound checking must be suppressed to allow the dual interpretation of variables as both integer and boolean (if anyone is offended by this objectionable programming trick, or if it fails to work under a particular implementation, I have a more portable version - but the price of portability is execution speed); similarly, overflow checking needs to be suppressed, but I have a remedy for this, too;
- iii) our implementation has whole word logical operations for and and or, but not for not - hence the use of 'acomp' and 'bcomp' (note - a subsequent release restricts and and or to boolean operations only)
- iv) the function must initially be passed a positive non-zero integer "seed" (parameter 'x'), and will thereafter update this seed and yield a real number in the range (0,1).

The feedback shift register method has been shown by Lewis [2] to give good results, and I have thoroughly tested this version with the usual statistical tests. (Pascal procedures for these tests are available from me).

Incidentally, I would be very pleased to hear from anyone else interested in Pascal and simulation.

Brian A. E. Meekings.

References.

1. Whittlesey, J.R.B. A comparison of the correlational behaviour of random number generators for the IBM 360. Comm ACM 11,9 (Sept 1968).
2. Lewis, T.G. Distribution Sampling for Computer Simulation. D.C.Heath and Co., Lexington and Toronto (1975)

```
function random (var x: integer): real;
```

```
const pshift = 2048;  
qshift = 16;  
big = 32767;
```

```
type dual = record  
  case dummy: boolean of  
    true: (i: integer);  
    false: (b: boolean)  
  and;
```

```
var a, b, acomp, bcomp: dual;
```

```
begin
```

```
(* exclusive or number and number shifted 4 places right *)  
a.i := x; b.i := a.i div qshift;  
acomp.i := big - a.i; bcomp.i := big - b.i;  
a.b := (a.b and bcomp.b) or (acomp.b and b.b);
```

```
(* exclusive or number and number shifted 11 places left *)  
b.i := a.i * pshift;  
acomp.i := big - a.i; bcomp.i := big - b.i;  
a.b := (a.b and bcomp.b) or (acomp.b and b.b);
```

```
(* convert to real result *)  
x := a.i;  
random := a.i / big  
end (* random *);
```

(* Jim Miner tried out the algorithm on a PDP8 (23 + 1 bit integers). He made following points: a) the results seemed to be better if the left-shift is circular; b) one has to be careful of multiply overflow; c) the exclusive-or's are more naturally expressed as set operations, and d) a seed of zero yields a constant zero result. His version is printed below. *)

```
FUNCTION RANDOM(VAR SEED: INTEGER) : REAL;  
CONST
```

```
PSHIFT = 65536; (* 2 ^ 16 *)  
PMOD = 128; (* 2 ^ 7 *)  
QSHIFT = 64; (* 2 ^ 6 *)  
MAXINT = 8388607; (* 2 ^ 23 - 1 *)
```

```
VAR
```

```
A, B:
```

```
RECORD CASE BOOLEAN OF  
  TRUE: (I: INTEGER);  
  FALSE: (S: PACKED SET OF 0..23);  
END;
```

```
BEGIN
```

```
A.I := ABS(SEED); B.I = A.I DIV QSHIFT; (*RIGHT SHIFT 6*)  
A.S := (A.S - B.S) + (B.S - A.S); (* XOR *)  
B.I := A.I MOD PMOD * PSHIFT + A.I DIV PMOD;  
(*LEFT SHIFT CIRCULAR 16*)  
A.S := (A.S - B.S) + (B.S - A.S);  
SEED := A.I; RANDOM := A.I / (1.0 + MAXINT)  
END (*RANDOM*)
```

A - 2 TimeLog

DOCUMENTATION : TIMELOG

Language : Pascal

Written : A.H.J. Sale
Thursday, 1978 March 2, 3.20pm

Use

To improve the quality of production PASCAL programs by making available a standard method of recording the date and time of a run.

User documentation

TimeLog is a Pascal procedure which writes on a globally declared file *output*, producing a single line which is a log-record of the date and time. It has no parameters, and is therefore used simply by including the text in the procedure declaration part of a program, and then activated by calling:

timeLog

The format of the printed line is chosen to avoid all the confusion created by numeric date and time information by conflicting American, English and European conventions; in addition a measure of redundancy is included by the weekday name. See date given above as an example.

Installation

The procedure will work without modification on Burroughs B6700/7700 installations using the University of Tasmania compiler. On other systems the machine-dependent part (identified clearly in the listing) will have to be altered to acquire the necessary information. (The B6700 pre-defined procedure *timestamp* puts year/month/day/hour/minute/second information into the array parameter elements, thereby avoiding any timing glitches of separate calls.) The lower-case letters and some other characters may have to be converted to suit some systems' lexical requirements. The procedure is easily modified to handle other Indo-European languages (e.g. French) by altering the text strings.

System documentation

The procedure is straightforward. Only a few things are worth noting.

- (i) Zeller's congruence is used to compute the weekday from the epoch.
- (ii) The date and time are written according to ISO standard format in descending order of significance (apart from the weekday).
- (iii) The minute value is printed without zero-suppression; other numeric codes are zero-suppressed, as is normal Pascal convention.

```
00010000
00010100
00010200
00010300
00010400
00010500
00010600
00010700
00010800
00010900
00011000
00011100
00011200
00011300
00011400
00011500
00011600
00011700
00011800
00011900
00020000
00021000
00022000
00023000
00024000
00025000
00026000
00027000
00028000
00029000
00030000
00031000
00032000
00033000
00034000
00035000
00036000
00037000
00038000
00039000
00040000
00041000
00042000
00043000
00044000
00045000
00046000
00047000
00048000
00049000
00050000
00051000
00052000
00053000
00054000
00055000
00056000
00057000
00058000
00059000
00060000
00061000
00062000
00063000
00064000
00065000
00066000
00067000
00068000
00069000
00070000
00071000
00072000
00073000
00074000
00075000
00076000
00077000
00078000
00079000
00080000
00081000
00082000
00083000

procedure timeLog;
  (#####)
(-----)
( This procedure prints out a basic log-record on the output
( file. It avoids the well-known problems of American and
( English date conventions, and the 24-hour clock confusion.
(-----)
var
  year      : 01..99;      ( two digits, 19xx assumed )
  month     : 1..12;      ( month number )
  day       : 1..31;      ( day in month )
  hour      : 0..23;      ( 24-hour clock assumed )
  minute    : 0..59;      ( minutes past the hour )

  epoch     : array[0..9] of integer;
              ( required for B6700 )

  adjyear   : 00..99;      ( Jan & Feb are taken as )
  adjmonth  : 1..12;      ( last months of prev year )
  weekday   : 0..6;       ( 0=Sunday, 1=Monday, etc )
  adjustedhour : 0..12;   ( conventional-clock )

begin
  ( The statements between here and the next comment should )
  ( be replaced by the equivalent for your system. Note the )
  ( range of the variables documented in the declarations. )
  timestampepoch;
  year :=epoch[0]-1900;
  month :=epoch[1];
  day :=epoch[2];
  hour :=epoch[3];
  minute :=epoch[4];
  ( this closes the machine-dependant part )

  ( compute the adjusted hour we use )
  adjustedhour:=hour mod 12;
  if (adjustedhour = 0) then adjustedhour:=12;

  ( adjust month and year information )
  if (month <= 2) then begin
    adjmonth:=month+10; adjyear:=year-1
  end else begin
    adjmonth:=month-2; adjyear:=year
  end;
  ( Zeller's congruence )
  weekday :=
    (((2 * adjmonth - 2) div 10) + day + adjyear +
     (adjyear div 4) + 1) mod 7;

  ( write the timeLog out )
  case weekday of
    0: write(output, 'Sunday');
    1: write(output, 'Monday');
    2: write(output, 'Tuesday');
    3: write(output, 'Wednesday');
    4: write(output, 'Thursday');
    5: write(output, 'Friday');
    6: write(output, 'Saturday');
  end;
  write(output, ', ', (year+1900):4, ' ');
  case month of
    1: write(output, 'January');
    2: write(output, 'February');
    3: write(output, 'March');
    4: write(output, 'April');
    5: write(output, 'May');
    6: write(output, 'June');
    7: write(output, 'July');
    8: write(output, 'August');
    9: write(output, 'September');
    10: write(output, 'October');
    11: write(output, 'November');
    12: write(output, 'December');
  end;
  write(output, ', day:2, ', adjustedhour:2, ':',
    (minute div 10):1, (minute mod 10):1);
  if (hour >= 12) then begin
    writein(output, ' PM. ')
  end else begin
    writein(output, ' AM. ')
  end;
end;
```

SOFTWARE TOOLS

S - 1 Compare

```

1 (*      COMPARE - Compare two text files and report their differences.
2 *
3 *      Copyright (C) 1977, 1978
4 *      James F. Miner
5 *      Social Science Research Facilities Center
6 *      University of Minnesota
7 *
8 *      General permission to make fair use in non-profit activities
9 *      of all or part of this material is granted provided that
10 *     this notice is given. To obtain permission for other uses
11 *     and/or machine readable copies write to:
12 *
13 *           The Director
14 *           Social Science Research Facilities Center
15 *           25 Blegen Hall
16 *           269 19th Ave. So.
17 *           University of Minnesota
18 *           Minneapolis, Minnesota 55455
19 *           U S A
20 *)
21
22
23 (*      Compare is used to display on "Output" the differences
24 *      between two similar texts ("Files" and "Fileb"). Notable
25 *      characteristics are:
26 *
27 *      - Compare is line oriented. The smallest unit of comparison
28 *      is the text line (ignoring trailing blanks). The present
29 *      implementation has a fixed maximum line length.
30 *
31 *      - By manipulating a program parameter, the user can affect
32 *      Compare's sensitivity to the "locality" of differences.
33 *      More specifically this parameter, "Minlinesformatch",
34 *      specifies the number of consecutive lines on each file
35 *      which must match in order that they be considered as
36 *      terminating the prior mismatch. A large value of
37 *      "Minlinesformatch" tends to produce fewer but larger
38 *      mismatches than does a small value. The value six appears
39 *      to give good results on Pascal source files but may be
40 *      inappropriate for other applications.
41 *
42 *      If compare is to be used as a general utility program,
43 *      "Minlinesformatch" should be treated as a program
44 *      parameter of some sort. It is declared as a constant here
45 *      for portability's sake.
46 *
47 *      - Compare employs a simple backtracking search algorithm to
48 *      isolate mismatches from their surrounding matches. This
49 *      requires (heap) storage roughly proportional the the size
50 *      of the largest mismatch, and time roughly proportional to
51 *      the square of the size of the mismatch for each mismatch.
52 *      For this reason it may not be feasible to use Compare on
53 *      files with very long mismatches.
54 *
55 *      - To the best of the author's knowledge, Compare utilizes
56 *      only features of Standard Pascal.
57 *)
58

```

```

59
60 program compare(filea, fileb, output);
61
62 const
63   version = '1.2p (78/03/01)';
64   linelength = 120;           { MAXIMUM SIGNIFICANT INPUT LINE LENGTH }
65   minlinesformatch = 6;      { NUMBER OF CONSECUTIVE EQUIVALENT }
66                               { LINES TO END A MIS-MATCH }
67
68 type
69   linepointer = ^line;
70   line =
71     packed record
72     nextline : linepointer;
73     length : 0..linelength;
74     image : packed array [1..linelength] of char
75   end;
76
77   stream =
78     record
79     cursor, head, tail : linepointer;
80     cursorlineno, headlineno, taillineno : integer;
81     endfile : boolean
82   end;
83
84 var
85   filea, fileb : text;
86   a, b : stream;
87   match : boolean;
88   endfile : boolean;         { SET IF END OF STREAM A OR B }
89
90   templine :
91     record
92     length : integer;
93     image : array [0..linelength] of char
94   end;
95
96   freelines : linepointer;   { FREE LIST OF LINE BUFFERS }
97
98   same : boolean;           { FALSE IF NO MIS-MATCHES OCCUR }
99
100
101 procedure comparefiles;
102
103 function endstream(var x : stream) : boolean;
104 begin { ENDSTREAM }
105   endstream := (x.cursor = nil) and x.endfile
106 end; { ENDSTREAM }
107
108 procedure mark(var x : stream);
109
110 { CAUSES BEGINNING OF STREAM TO BE POSITIONED BEFORE }
111 { CURRENT STREAM CURSOR. BUFFERS GET RECLAIMED, LINE }
112 { COUNTERS RESET, ETC. }
113
114 var
115   p : linepointer;
116
117 begin { MARK }
118   with x do
119     if head <> nil then
120       begin

```

```

121   while head <> cursor do ( RECLAIM BUFFERS )
122     begin
123       with head^ do
124         begin p := nextline;
125         nextline := freelines; freelines := head
126         end;
127         head := p
128         end;
129         headlineno := cursorlineno;
130         if cursor = nil then
131           begin tail := nil; taillineno := cursorlineno end
132         end
133     end; ( MARK )
134
135 procedure movecursor(var x : stream; var filex : text);
136
137   ( FILE IS THE INPUT FILE ASSOCIATED WITH STREAM X. THE )
138   ( CURSOR FOR X IS MOVED FORWARD ONE LINE, READING FROM X )
139   ( IF NECESSARY, AND INCREMENTING THE LINE COUNT. ENDFILE )
140   ( IS SET IF EOF IS ENCOUNTERED ON EITHER STREAM. )
141
142 procedure readline;
143 var
144   newline : linepointer;
145   c, c2 : 0..linelength;
146   begin ( READLINE )
147     if not x.endfile then
148       begin
149         c := 0;
150         while not eoln(filex) and (c < linelength) do
151           begin c := c + 1; templine.image[c] := filex^; get(filex) end;
152         readln(filex);
153         while templine.image[c] = ' ' do c := c - 1;
154         if c < templine.length then
155           for c2 := c+1 to templine.length do templine.image[c2] := ' ';
156         templine.length := c;
157         newline := freelines;
158         if newline = nil then new(newline)
159         else freelines := freelines^.nextline;
160         pack(templine.image, 1, newline^.image);
161         newline^.length := c;
162         newline^.nextline := nil;
163         if x.tail = nil then
164           begin x.head := newline;
165             x.taillineno := 1; x.headlineno := 1
166           end
167         else
168           begin x.tail^.nextline := newline;
169             x.taillineno := x.taillineno + 1
170           end;
171         x.tail := newline;
172         x.endfile := eof(filex);
173       end
174     end; ( READLINE )
175
176   begin ( MOVECURSOR )
177     if x.cursor <> nil then
178       begin
179         if x.cursor = x.tail then readline;
180         x.cursor := x.cursor^.nextline;
181         if x.cursor = nil then endfile := true;
182         x.cursorlineno := x.cursorlineno + 1

```

```

183     end
184   else
185     if not x.endfile then ( BEGINNING OF STREAM )
186       begin
187         readline; x.cursor := x.head;
188         x.cursorlineno := x.headlineno
189       end
190     else ( END OF STREAM )
191       endfile := true;
192   end; ( MOVECURSOR )
193
194 procedure backtrack(var x : stream; var xlines : integer);
195
196   ( CAUSES THE CURRENT POSITION OF STREAM X TO BECOME THAT )
197   ( OF THE LAST MARK OPERATION. I.E., THE CURRENT LINE )
198   ( WHEN THE STREAM WAS MARKED LAST BECOMES THE NEW CURSOR. )
199   ( XLINES IS SET TO THE NUMBER OF LINES FROM THE NEW CURSOR )
200   ( TO THE OLD CURSOR, INCLUSIVE. )
201
202   begin ( BACKTRACK )
203     xlines := x.cursorlineno + 1 - x.headlineno;
204     x.cursor := x.head; x.cursorlineno := x.headlineno;
205     endfile := endstream(a) or endstream(b)
206   end; ( BACKTRACK )
207
208 procedure comparelines(var match : boolean);
209
210   ( COMPARE THE CURRENT LINES OF STREAMS A AND B, RETURNING )
211   ( MATCH TO SIGNAL THEIR (NON-) EQUIVALENCE. EOF ON BOTH STREAMS )
212   ( IS CONSIDERED A MATCH, BUT EOF ON ONLY ONE STREAM IS A MISMATCH )
213
214   begin ( COMPARELINES )
215     if (a.cursor = nil) or (b.cursor = nil) then
216       match := endstream(a) and endstream(b)
217     else
218       begin
219         match := (a.cursor^.length = b.cursor^.length);
220         if match then
221           match := (a.cursor^.image = b.cursor^.image)
222         end
223       end; ( COMPARELINES )
224
225 procedure findmismatch;
226   begin ( FINDMISMATCH )
227     ( NOT ENDFILE AND MATCH )
228     repeat ( COMPARENEXTLINES )
229       movecursor(a, filea); movecursor(b, fileb);
230       mark(a); mark(b);
231       comparelines(match)
232     until endfile or not match;
233   end; ( FINDMISMATCH )
234
235 procedure findmatch;
236   var
237     advanceb : boolean; ( TOGGLE ONE-LINE LOOKAHEAD BETWEEN STREAMS )
238
239   procedure search(var x : stream; ( STREAM TO SEARCH )
240     var filex : text;
241     var y : stream; ( STREAM TO LOOKAHEAD )
242     var filey : text);
243
244     ( LOOK AHEAD ONE LINE ON STREAM Y, AND SEARCH FOR THAT LINE )
245     ( BACKTRACKING ON STREAM X. )

```



```

246
247 var
248   count : integer; ( NUMBER OF LINES BACKTRACKED ON X )
249
250 procedure checkfullmatch;
251 ( FROM THE CURRENT POSITIONS IN X AND Y, WHICH MATCH, )
252 ( MAKE SURE THAT THE NEXT MINLINESFORMATCH-1 LINES ALSO )
253 ( MATCH, OR ELSE SET MATCH := FALSE. )
254 var
255   n : integer;
256   savexcur, saveycur : linepointer;
257   savexline, saveyline : integer;
258 begin ( CHECKFULLMATCH )
259   savexcur := x.cursor; saveycur := y.cursor;
260   savexline := x.cursorlineno; saveyline := y.cursorlineno;
261   comparelines(match);
262   n := minlinesformatch - 1;
263   while match and (n <> 0) do
264     begin movecursor(x, filex); movecursor(y, filey);
265     comparelines(match); n := n - 1
266     end;
267   x.cursor := savexcur; x.cursorlineno := savexline;
268   y.cursor := saveycur; y.cursorlineno := saveyline;
269 end; ( CHECKFULLMATCH )
270
271 begin ( SEARCH )
272   movecursor(y, filey); backtrack(x, count);
273   checkfullmatch; count := count - 1;
274   while (count <> 0) and not match do
275     begin
276       movecursor(x, filex); count := count - 1;
277       checkfullmatch
278     end
279 end; ( SEARCH )
280
281 procedure printmismatch;
282 var
283   emptya, emptyb : boolean;
284
285 procedure writetext(p, q : linepointer);
286 begin ( WRITETEXT )
287   writeln;
288   while (p <> nil) and (p <> q) do
289     begin write(' ');
290     if p^.length = 0 then writeln
291     else writeln(p^.image : p^.length);
292     p := p^.nextline
293     end;
294   if p = nil then writeln(' *** eof ***');
295   writeln
296 end; ( WRITETEXT )
297
298 procedure writelineno(var x : stream);
299 var
300   f, l : integer;
301 begin ( WRITELINENO )
302   f := x.headlineno; l := x.cursorlineno - 1;
303   write('line');
304   if f = l then write(' ', f:l)
305   else write('s ', f:l, ' to ', l:l);
306   if x.cursor = nil then write(' (before eof)');
307 end; ( WRITELINENO )
308

```

```

309
310 procedure printextratext(var x : stream; xname : char;
311 var y : stream; yname : char);
312 begin ( PRINTEXTRATEXT )
313   write(' extra text on file', xname, ', ');
314   writelineno(x); writeln;
315   if y.head = nil then
316     writeln(' before eof on file', yname)
317   else
318     writeln(' between lines ', y.headlineno-1:l, ' and ',
319             y.headlineno:l, ' of file', yname);
320   writetext(x.head, x.cursor)
321 end; ( PRINTEXTRATEXT )
322
323 begin ( PRINTMISMATCH )
324   writeln(' *****');
325   emptya := (a.head = a.cursor);
326   emptyb := (b.head = b.cursor);
327   if emptya or emptyb then
328     if emptya then printextratext(b, 'b', a, 'a')
329     else printextratext(a, 'a', b, 'b')
330   else
331     begin
332       writeln(' mismatch:'); writeln;
333       write(' files, '); writelineno(a); writeln(':');
334       writetext(a.head, a.cursor);
335       write(' fileb, '); writelineno(b); writeln(':');
336       writetext(b.head, b.cursor)
337     end
338   end; ( PRINTMISMATCH )
339
340 begin ( FINDMATCH )
341 ( NOT MATCH )
342   advanceb := true;
343 repeat
344   if not endfile then advanceb := not advanceb
345   else advanceb := endstream(a);
346   if advanceb then search(a, filea, b, fileb)
347   else search(b, fileb, a, filea)
348 until match;
349   printmismatch;
350 end; ( FINDMATCH )
351
352 begin ( COMPAREFILES )
353   match := true; ( I.E., BEGINNINGS-OF-FILES MATCH )
354 repeat
355   if match then findmismatch else begin same := false; findmatch and
356   until endfile and match;
357   ( MARK(A); MARK(B); MARK END OF FILES, THEREBY DISPOSING BUFFERS )
358 end; ( COMPAREFILES )
359
360 procedure initialize;
361
362 procedure initstream(var x : stream; var filex : text);
363 begin ( INITSTREAM )
364   with x do
365     begin
366       cursor := nil; head := nil; tail := nil;
367       cursorlineno := 0; headlineno := 0; taillineno := 0
368     end;
369   reset(filex); x.endfile := eof(filex);
370 end; ( INITSTREAM )
371

```

```

371
372 begin ( INITIALIZE )
373   initstream(a, filea); initstream(b, fileb);
374   endfile := a.endfile or b.endfile;
375   freelines := nil;
376   templn.length := linal.length;
377   templn.image[0] := 'x'; { SENTINEL }
378   end; {INITIALIZE}
379
380
381 begin (COMPARE)
382   initialize;
383   page(output);
384   writeln('      compare. version ', version);
385   writeln;
386   writeln(' match criterion = ', minlinesformatch:1, ' lines. ');
387   writeln;
388   if a.endfile then writeln(' filea is empty. ');
389   if b.endfile then writeln(' fileb is empty. ');
390   if not endfile then
391     begin same := true;
392     comparefiles;
393     if same then writeln(' no differences. ');
394   end
395 end. { COMPARE }

```

(* The following output from Compare was generated on a CDC Cyber 74. The data used was the source text of the Compare program itself, modified in 3 places by performing a change, a deletion, and an insertion. The original source is "File A". *)

```

COMPARE. VERSION 1.2P (78/03/01)

MATCH CRITERION = 6 LINES.

*****
MISMATCH:

FILEA, LINE 101:

* PROCEDURE COMPAREFILES;

FILEB, LINE 101:

* PROCEDURE COMPAREFOOLS;

*****
EXTRA TEXT ON FILEA, LINE 131
BETWEEN LINES 130 AND 131 OF FILEB

* BEGIN TAIL := NIL; TAILLINEO := CURSORLINEO END

*****
EXTRA TEXT ON FILEB, LINE 162
BETWEEN LINES 162 AND 163 OF FILEA

* GARBAGE;

```

PERFORMANCE MEASUREMENT OF PASCAL PROGRAMS
USING AUGMENT AND ANALYZE

- Andy Mickel 78/03/14.
University Computer Center
University of Minnesota
Minneapolis, MN 55455

What AUGMENT and ANALYZE Do

Suppose you want to examine the execution efficiency of your Pascal program--perhaps to make improvements to those parts which take the most computer time.

AUGMENT and ANALYZE are designed to obtain rough measures of such execution times, particularly for large Pascal programs. Unlike other kinds of performance measurement, AUGMENT and ANALYZE assume the PROCEDURE and FUNCTION to be the smallest unit of a program to be monitored. This is a satisfactory assumption because well-written Pascal programs produced by stepwise refinement naturally are composed of proper-sized procedures and functions.

The general principle used by these programs is that the value of the non-standard Pascal function CLOCK (which returns the elapsed processing time in milliseconds) can be sampled at procedure or function entry and exit. When the expression (exittime - entrytime), is evaluated the time spent within the particular procedure or function can be ascertained.

AUGMENT is the program which inserts the necessary CLOCK-sampling code into your Pascal source program for every procedure and function entry and exit. It thus causes your program to capture timing information and to write it out to a file.

Next you compile and execute your program, which actually produces the file of dynamic timing measurements.

ANALYZE then reads the timing file produced, and writes a report, which gives the name of each procedure or function, the number of times it was called, and the execution time it consumed for all calls and per call.

AUGMENT and ANALYZE therefore provide a nearly machine-independent method for gathering performance-measurement data about a Pascal program. Most Pascal implementations have the required CLOCK function which returns the elapsed processor time in milliseconds.

It is sometimes necessary to exclude the monitoring of excessively called procedures and functions in large programs. A feature of AUGMENT allows you to specify any number of names to be excluded.

How to Use AUGMENT and ANALYZE

Under CDC 6000/Cyber 70,170 operating systems, AUGMENT and ANALYZE are control statements. Thus the following 3 batch commands do the job:

```

AUGMENT(the file name of your Pascal source program)
PASCAL(INTER/L-,G+)
ANALYZE.

```

The program headings for AUGMENT and ANALYZE are:

```
AUGMENT(INPUT, EXCEPT, INTER, INTER2, OUTPUT)
```

where:

INPUT is the textfile containing the Pascal source program to be AUGMENTed.

EXCEPT is the textfile containing a list of names (one to a line with no leading blanks) of procedures and functions to be excluded from measurement. EXCEPT can be an empty file in which case no procedures or functions will be excluded.

INTER is the textfile on which the AUGMENTED version of the Pascal source program is written.

INTER2 is the binary file on which only the names of each procedure and function in the Pascal source program is written for use by ANALYZE.

OUTPUT is the textfile on which error messages are written if problems occur during AUGMENTING. A report is written on OUTPUT verifying which procedures or functions were excluded, if any.

The error messages are:

*TOO MANY PROCEDURES AND FUNCTIONS TO AUGMENT.

(A limit of 2000 is imposed.)

*"BEGIN" EXPECTED.

*"END" EXPECTED.

(There's something wrong with the statement part of the Pascal source program which is being AUGMENTed; it began with some reserved symbol other than "begin" or there weren't enough "END"s to match "BEGIN"s.)

*"PROGRAM" EXPECTED.

(AUGMENT couldn't find "PROGRAM" as the first reserved symbol in the Pascal source program. Possibly the INPUT file was empty.)

*UNDECLARED LABEL.

AUGMENT couldn't find a label referred to by a GOTO statement.

ANALYZE(OUTPUT, INTER2, TIMING)

where:

OUTPUT is the textfile on which the performance measurement report is written or alternatively the error message: *TIMING FILE EMPTY.

INTER2 is the binary file on which the names of each procedure and function in the Pascal source program was written by AUGMENT.

TIMING is the binary file containing the dynamic timing measurements resulting from execution of the AUGMENTED Pascal program.

Note: The identifier "TIMING" is added to the Pascal source program by AUGMENT and must not appear in any procedure or function which is to be monitored. When you use AUGMENT and ANALYZE, it is probably a good idea to consider the file names INTER, INTER2, and TIMING reserved.

In summary, there are four steps to the performance measurement process:

- 1) [Pascal source program] -> AUGMENT -> INTER and INTER2

- 2) INTER -> PASCAL Compiler -> [Pascal binary program]

- 3) [input data for Pascal program] -> Pascal binary program -> TIMING and [results
***** from Pascal program]
- 4) TIMING and INTER2 -> ANALYZE -> [performance measurement report]

EXAMPLE

Below are a test program, its AUGMENTED version, and the performance measurement report:

The source of the test program:

```
PROGRAM TEST(OUTPUT);
LABEL 5;
VAR N: INTEGER;

PROCEDURE A;

PROCEDURE B;
BEGIN N := N + 1;
IF ODD(N DIV 2) THEN A ELSE B
END (*B*);

BEGIN (*A*)
N := N + 1;
IF N > 200 THEN GOTO 5;
B
END (*A*);
BEGIN N := 0; A; 5: END.
```

The AUGMENTED version of the test program:

```
PROGRAM TEST(OUTPUT,TIMING);
LABEL 5;
VAR
TIMING:FILE OF PACKED RECORD I:0..2000;T:0..99999999;M:0..2 END;
N: INTEGER;

PROCEDURE A;

PROCEDURE B;
BEGIN
WITH TIMING^ DO BEGIN I:= 3;T:=CLOCK;M:=0 END;PUT(TIMING);
N := N + 1;
IF ODD(N DIV 2) THEN A ELSE B
;
WITH TIMING^ DO BEGIN I:= 3;T:=CLOCK;M:=1 END;PUT(TIMING)
END
(*B*);

BEGIN
WITH TIMING^ DO BEGIN I:= 1;T:=CLOCK;M:=0 END;PUT(TIMING);
(*A*)
N := N + 1;
IF N > 200 THEN BEGIN
WITH TIMING^ DO BEGIN I:= 2;T:=CLOCK;M:=2 END;PUT(TIMING);
GOTO 5 END
;
B
;
WITH TIMING^ DO BEGIN I:= 2;T:=CLOCK;M:=1 END;PUT(TIMING)
END
(*A*);
BEGIN REWRITE(TIMING);
WITH TIMING^ DO BEGIN I:= 1;T:=CLOCK;M:=0 END;PUT(TIMING);
N := 0; A; 5: ;
WITH TIMING^ DO BEGIN I:= 1;T:=CLOCK;M:=1 END;PUT(TIMING)
END
.
```

The report from ANALYZE:

PERFORMANCE MEASUREMENT SUMMARY FOR PASCAL PROGRAM: TEST

MODULE NAME	CALLS		EXECUTION TIME (MILLISECONDS)		
	TIMES CALLED	PERCENT OF TOTAL	AVERAGE PER CALL	MODULE TOTAL	PERCENT OF TOTAL
A	52	25.490	0.15	8	27.586
B	151	74.020	0.13	20	68.966
TEST	1	0.490	1.00	1	3.448
-----	-----	-----	-----	-----	-----
TOTALS	204	100.000	0.14	29	100.000

From the summary provided by AUGMENT and ANALYZE, you can identify which procedures and functions to improve for greater execution efficiency. In general, it pays to concentrate on procedures and functions which are frequently called and take a significant amount of the execution time of the total program. Procedures and functions which have a large average execution time per call, but which are only called a few times are not worth worrying about.

If one or more procedures or functions seem to dominate the results, it might be a good idea to monitor the program with these modules excluded from measurement. Use the except feature provided by AUGMENT.

History

AUGMENT and ANALYZE were conceived originally under the names PROFILE and PRINRES in 1975-1976 by S. Matwin and M. Missala, of the Polish Academy of Sciences Computer Centre, PKIN, Warwaw, Poland. The goal of the project was to build a simple tool to measure very large programs -- such as the Pascal compiler itself. A paper describing their successful work entitled: "A Simple, Machine Independent Tool for Obtaining Rough Measures of Pascal Programs," appeared in SIGPLAN Notices (11:8) August, 1976, pages 42-45.

Their successful implementation was on CDC machines using Pascal-6000. In 1976, Richard J. Cichelli of Lehigh University Mathematics Department and the American Newspaper Publishers Association Research Institute, obtained the programs and documented and distributed them to the Pascal community in the United States.

In 1977, Herb Rubenstein and Andy Mickel of the University of Minnesota Computer Center, modified the programs for coding style and to increase portability, fixed bugs, and improved the performance of the programs themselves. We also removed several limitations (the built-in restrictions regarding the use of non-local GOTOs within procedures and functions as well as the monitoring of procedures named NEXTCH).

The programs are now supported with the Pascal-6000 system which is distributed to CDC installations around the world.

(* Note: In the programs listings following, empty comments denote lines with possible or outright machine dependencies. *)

```
1 [* AUGMENT - AUGMENT PASCAL PROGRAMS WITH CODE TO GATHER
2 * EXECUTION TIME PERFORMANCE MEASUREMENTS.
3 *
4 * S. MATWIN AND
5 * M. MISSALA 1975.
6 * POLISH ACADEMY OF SCIENCES COMPUTER CENTRE.
7 * PKIN, WARSAW POLAND.
8 *
9 * REFERENCE: "A SIMPLE MACHINE INDEPENDENT
10 * TOOL FOR OBTAINING ROUGH MEASURES
11 * OF PASCAL PROGRAMS."
12 * SIGPLAN NOTICES, 1976 AUGUST, PP. 42-45.
13 *
14 * MODIFIED, GENERALIZED, AND RENAMED
15 * FROM "PROFILE" TO "AUGMENT" BY:
16 * A. B. MICKEL 77/08/04.
17 * H. U. RUBENSTEIN 77/06/01.
18 * UNIVERSITY OF MINNESOTA COMPUTER CENTER
19 * MINNEAPOLIS, MN 55455 USA.
20 *
21 * THE NAMES AND ORGANIZATIONS GIVEN HERE MUST NOT BE
22 * DELETED IN ANY USE OF THIS PROGRAM.
23 *
24 * SEE THE PTOOLS WRITEUP (UNDER MEASURE) FOR
25 * EXTERNAL DOCUMENTATION.
26 *
27 * AUGMENT (INTERNAL DOCUMENTATION).
28 *
29 ** AUGMENT INSERTS CODE TO CREATE A TIMING FILE IN THE PROGRAM
30 * HEADER, DECLARATION PART, AND STATEMENT PART OF THE PROGRAM
31 * TO BE MONITORED. CODE IS ALSO INSERTED IN THE STATEMENT
32 * PART OF EACH PROCEDURE AND FUNCTION TO WRITE CLOCK
33 * MEASUREMENTS (AT ENTRY, EXIT, OR GOTOENTRY) TO THE TIMING
34 * FILE WHEN THE PROGRAM IS EXECUTED.
35 *
36 * AUGMENT MUST PARSE A SUBSET OF PASCAL AND THEREFORE HAS A
37 * LEXICAL ANALYZER. THE TIMING FILE IS PROCESSED BY THE
38 * COMPANION PROGRAM CALLED ANALYZE.
39 *
40 *
41 )
42
43
44 { $R-,T-,P-,U+ }
45
46
47 program augment(input, except, inter, inter2, output);
48
49 label
50 13 (EXIT FOR PROGRAM ERRORS);
51
52 const
53     beginsy = 1;
54     casesy = 2;
55     endsy = 3;
56 {}
57 {}
58 {}
59     externsy = 4;
60     fortransy = 5;
61     forwardy = 6;
62     funcsy = 7;
63     gotosy = 8;
64     labelsy = 9;
65     procsy = 10;
66     programsy = 11;
67     varsy = 12;
```

```

65 maxmodules = 2000;
66 llmax = 120 { LINE LENGTH MAX };
67 llmin = 72 { LINE LENGTH MIN };
68 alfa leng = 10;
69
70 type
71     alfa = packed array [1 .. alfa leng] of char;
72     codetype = (entry, exit, gotoentry, declare);
73     symbols = beginsy .. varsy;
74     namenode = record
75         name: alfa;
76         link: ^ namenode
77     end;
78     modulecnt = 0 .. maxmodules;
79     labelptr = ^ labelnode;
80     labelnode = record
81         labl: 0 .. 9999;
82         declaredin: modulecnt;
83         next: labelptr
84     end { LABELNODE };
85
86 var
87     idlen,           { IDENTIFIER LENGTH }
88     lastidlen: 0 .. alfa leng;
89     sy: symbols;
90     chbuf: array [1 .. alfa leng] of char;
91     identifier: alfa;
92     number: 0 .. 9999;
93     key: array [symbols] of alfa;
94     badnames,
95     readinglabels: boolean;
96     linelength,
97     colcnt: integer;
98     ch: char;
99     inter: text      { AUGMENTED PROGRAM FILE };
100    except: text     { FILE OF EXCEPTED MODULE NAMES };
101    inter2: file of alfa { FILE OF ALL MODULE NAMES };
102    badlist: ^ namenode { LIST OF EXCEPTED MODULE NAMES };
103    count: modulecnt { RUNNING COUNT OF MODULES };
104
105
106 procedure nextch;
107
108 begin
109     get(input);
110     ch := input;
111     colcnt := colcnt + 1;
112     while (not eoln(input) and (colcnt > linelength)) do
113         get(input);
114         if eoln(input) then
115             colcnt := 0
116         end { NEXTCH };
117
118
119 procedure advance;
120
121 begin
122     if eoln(input)
123     then
124         writeln(inter)
125     else
126         write(inter, ch);
127     nextch
128 end { ADVANCE };
129
130

```

```

131 procedure readid;
132 { GLOBAL : CHBUF, CH, IDENTIFIER, IDLEN, LASTIDLEN }
133
134 begin { READID }
135     idlen := 0;
136     repeat
137         if idlen < alfa leng then
138             begin
139                 idlen := idlen + 1;
140                 chbuf[idlen] := ch
141             end { IF };
142         nextch
143     until not (ch in ['a' .. 'z', '0' .. '9']);
144     if idlen >= lastidlen
145     then
146         lastidlen := idlen
147     else
148         repeat
149             chbuf[lastidlen] := ' ';
150             lastidlen := lastidlen - 1
151         until lastidlen = idlen;
152     pack(chbuf, 1, identifier)
153 end { READID };
154
155
156 procedure writeid;
157
158 var
159     { GLOBAL : CHBUF, IDLEN }
160     i: integer;
161
162 begin { WRITEID }
163     i := 1;
164     while i <= idlen do
165         begin
166             write(inter, chbuf[i]);
167             i := i + 1
168         end { WHILE }
169     end { WRITEID };
170
171
172 procedure comment;
173
174 begin
175     advance;
176     repeat
177         while ch <> ';' do
178             advance;
179         advance
180     until ch = ';';
181     advance
182 end { COMMENT };
183
184
185 procedure stdcomment;
186
187 begin
188     repeat
189         advance
190     until ch = ';';
191     advance
192 end { STDCOMMENT };
193
194
195 procedure scan;
196

```

```

197 ( FIND NEXT IDENTIFIER (OR NUMBER IF READINGLABELS).
198   SKIP STRINGS AND COMMENTS. )
199
200   label
201   21;
202
203 ( GLOBAL : IDENTIFIER,CH,NUMBER,BADNAMES,READINGLABELS )
204
205
206   function nokey(id: alfa): boolean;
207
208   var
209   { GLOBAL : SY,KEY }
210   i, j: integer;
211
212   begin { BINARY SEARCH }
213   i := beginsy;
214   j := varsy;
215   repeat
216   sy := (i + j) div 2;
217   if key[sy] <= id then
218     i := sy + 1;
219   if key[sy] >= id then
220     j := sy - 1;
221   until i > j;
222   nokey := key[sy] <> id
223   end { NOKEY };
224
225
226   begin { SCAN }
227   while not eof(input) do
228     begin
229     while not eoln(input) do
230       begin
231       if ch = ' '
232       then
233         advance
234       else
235       if ch in ['a' .. 'z']
236       then
237         begin
238         readid;
239         readinglabels := false;
240         if nokey(identifier) and not badnames
241         then
242           writeid
243         else
244           goto 21 { EXIT ON KEY OR EXPECTED ID }
245         end {IF}
246       else
247       if ch in ['0' .. '9']
248       then
249         if readinglabels
250         then
251           begin
252           read(number);
253           ch := input;
254           goto 21 { EXIT ON LABEL }
255         end {IF}
256       else
257         repeat
258         advance
259         until not (ch in ['a' .. 'z', '0' .. '9'])
260       else
261       if ch = ''
262       then

```

```

263   begin
264     repeat
265     advance
266     until ch = '' ;
267     advance
268     end {IF}
269   else
270     if ch = '('
271     then
272       begin
273       advance;
274       if ch = '#' then
275         comment
276       end {IF}
277     else
278       if ch = '('
279       then
280         stdcomment
281       else
282         advance
283       end {WHILE};
284       writeIn(inter);
285       nextch
286     end {WHILE};
287   21:
288   end { SCAN };
289
290
291   procedure complmodule(lastl: labelptr);
292
293   { PROCESS THE BLOCK OF A PROGRAM, PROCEDURE, OR FUNCTION TO FIND
294     THE APPROPRIATE CODE INSERTION POINTS. LASTL IS THE HEAD OF THE
295     LIST OF LABELS WHOSE SCOPE APPLIES TO THE BLOCK. COMPLMODULE
296     MUST PARSE LABEL, VAR, PROCEDURE, AND FUNCTION DECLARATIONS, AS
297     WELL AS GOTO STATEMENTS AND THE COMPOUND STATEMENT FORMING THE
298     STATEMENT PART OF EACH MODULE. }
299
300   var
301   { GLOBAL : IDENTIFIER,KEY,SY,CH,READINGLABELS,NUMBER,COUNT }
302   name: alfa;
303   depth: integer;
304   paraas: boolean;
305   l: labelptr;
306   gotolabel: 0 .. 9999;
307   looking: boolean;
308   tag: modulecnt;
309
310
311   procedure insertnewtext(code: codetype);
312
313   begin
314     case code of
315     entry:
316       begin
317       write(inter, 'with timing' do begin ');
318       write(inter, 'i:=', tag:4, ',');
319       write(inter, 't:=clock;a:=0' {ENTRY});
320       writeln(inter, ' end;put(timing)');
321     end {ENTRY};
322   exit:
323     begin
324     writeln(inter, ',');
325     write(inter, 'with timing' do begin ');
326     write(inter, 'i:=', tag:4, ',');
327     write(inter, 't:=clock;a:=1' {EXIT});
328     writeln(inter, ' end;put(timing)');

```

```

329     end (EXIT);
330 gotoentry:
331     begin
332         write(inter, 'with timing" do begin ');
333         write(inter, 'i:=', l^.declaredin:4, ',');
334 {}         write(inter, 't:=clock;m:=2' (GOTOENTRY));
335         writeln(inter, ' end;put(timing);');
336     end (GOTOENTRY);
337 declare:
338     begin
339         writeln(inter, 'var ');
340         write(inter, 'timing:file of packed record ');
341         write(inter, 'i:0..2000;');
342         write(inter, 't:0..99999999;');
343         write(inter, 'm:0..2');
344         writeln(inter, ' end;');
345     end (DECLARE)
346 end { CASE CODE OF }
347 end { INSERTNEWTEXT };
348
349
350 function nameok: boolean;
351
352 { CHECK PROCEDURE OR FUNCTION NAME AGAINST LIST OF NAMES TO BE
353   EXCLUDED. }
354
355 var
356     { GLOBAL : BADLIST,NAME }
357     n : ^namenode;
358     looking: boolean;
359
360 begin { NAMEOK }
361     n := badlist;
362     looking := true;
363     while (n <> nil) and looking do
364         begin
365             looking := n^.name <> name;
366             n := n^.link
367         end (WHILE);
368     nameok := looking
369 end {NAMEOK};
370
371
372 begin (COMPLMODULE)
373 {} while not (ch in ['a' .. 'z']) do
374     if ch = '('
375     then
376         begin
377             advance;
378             if ch = '*' then
379                 comment
380             end
381         else
382             if ch = '{'
383             then
384                 stdcomment
385             else
386                 advance;
387         readid;
388         name := identifier;
389         writeid;
390         tag := count;
391         params := false;
392         while not params and (ch <> ';') do
393             if ch = '('
394             then

```

```

395         begin
396             advance;
397             if ch = '*'
398             then
399                 comment
400             else
401                 params := true
402             end (IF)
403         else
404             if ch = '{'
405             then
406                 stdcomment
407             else
408                 advance;
409         if params
410         then
411             while ch <> ')' do { READ THROUGH PARAMETER LIST }
412                 if ch = '{'
413                 then
414                     stdcomment
415                 else
416                     if ch = '('
417                     then
418                         begin
419                             advance;
420                             if ch = '*' then
421                                 comment
422                             end (IF)
423                         else
424                             advance;
425                     if tag = 1 { MAIN PROGRAM }
426                     then
427                         write(inter, ',timing')
428                     else
429                         write(inter, ch);
430                     nextch;
431                     scan;
432                 {}
433             {}
434             {}
435             {}
436             {}
437         else
438             begin
439                 count := count + 1;
440                 if count = maxmodules then
441                     begin
442                         writeln(' *too many procedures and',
443                             ' functions to augment. ');
444                         goto 13
445                     end;
446                 write(inter2, name);
447                 if sy = labelsy { LABEL DECLARATION }
448                 then
449                     begin { READ LOCAL LABELS }
450                         writeid;
451                         readinglabels := true;
452                         scan;
453                         repeat
454                             new(l);
455                             l^.labl := number;
456                             l^.declaredin := tag;
457                             write(inter, number: 1);
458                             l^.next := lastl;
459                             lastl := l;
460                             scan
461                         until not readinglabels

```

```

461     end (IF);
462
463   while sy in [casesy, endsy] do ( TYPE DECLARATION )
464     begin
465       writeid;
466       scan
467     end (WHILE);
468   if tag = 1 ( MAIN PROGRAM )
469   then
470     insertnewtext(declare)
471   else
472     if not (sy in [beginsy, funcsy, procsy]) then
473       writeid;
474
475   if sy = varsy
476   then
477     begin
478       scan;
479       while sy in [casesy, endsy] do
480         begin (CASESY,ENDSY IN AN ANONYMOUS TYPE )
481           writeid;
482           scan
483         end (WHILE)
484       end (IF);
485
486   while sy in [funcsy, procsy] do
487     begin
488       writeid;
489       complmodule(lastl)
490     end (WHILE);
491
492   if sy = beginsy ( STATEMENT PART )
493   then
494     begin
495       depth := 1;
496       writeid;
497       if tag = 1 ( MAIN PROGRAM )
498       then
499         writeln(inter, ' rewrite(timing);')
500       else
501         writeln(inter);
502       if nameok then
503         insertnewtext(entry)
504       end (IF)
505     else
506       begin
507         writeln(' ''begin'' expected. ');
508         goto l3
509       end (ELSE);
510
511   repeat ( LOOK FOR LAST ENDSY )
512     scan;
513     if sy = gotosy
514     then
515       begin ( CHECK AGAINST LOCAL LABELS )
516         readinglabels := true;
517         scan;
518         gotolabel := number;
519         readinglabels := false;
520         looking := true;
521         l := lastl;
522         while (l <> nil) and looking do
523           if l^.labl = gotolabel
524           then
525             looking := false
526           else

```

```

527     l := l^.next;
528   if looking
529   then
530     begin
531       writeln(' *undeclared label ', gotolabel: 1);
532       goto l3
533     end (IF)
534   else
535     begin
536       if l^.declaredin <> tag then
537         begin ( EXIT GOTO )
538           writeln(inter, 'begin');
539           if nameok then
540             insertnewtext(gotoentry)
541           end (IF);
542           write(inter, 'goto ', gotolabel: 1);
543           if l^.declaredin <> tag then
544             writeln(inter, 'end')
545           end (ELSE)
546         end (IF)
547       else
548         if sy in [beginsy, casesy]
549         then
550           begin
551             depth := depth + 1;
552             writeid
553           end (IF)
554         else
555           if sy = endsy
556           then
557             begin
558               depth := depth - 1;
559               if depth <> 0 then
560                 writeid
561               end (IF)
562             else
563               begin
564                 writeln(' ''end'' expected. ');
565                 goto l3
566               end (ELSE)
567             until depth = 0;
568             if nameok then
569               insertnewtext(exit);
570             writeln(inter, 'end');
571             end (ELSE);
572             scan
573           end (COMPLMODULE);
574
575   procedure readbadnames;
576
577   var
578     { GLOBAL : BADLIST,CHBUF }
579     n: ^namenode;
580     i: 1 .. alfa leng;
581
582   begin ( READBADNAMES )
583     writeln('the following procedures will not be augmented' :50);
584     writeln;
585     repeat
586       new(n);
587       for i := 1 to alfa leng do
588         if eoln(except)
589         then
590           chbuff[i] := '

```



```

593     else
594         read(except, chbuf[1]);
595     readln(except);
596     pseek(chbuf, 1, n.name);
597     n.link := badlist;
598     badlist := n;
599     writeln(n.name :25);
600     writeln
601 until eof(except)
602 and ( READBADNAMES );
603
604
605 begin ( MAIN PROGRAM )
606     rewrite(inter);
607     rewrite(inter2);
608     reset(except);
609     oh := input";
610     count := 1;
611     colont := 1;
612     linelength := lmax;
613     lastidlen := alfaleng;
614 {} linelimit(inter, maxint);
615 key[begin] := 'begin';
616 key[case] := 'case';
617 key[end] := 'end';
618 {} key[extern] := 'extern';
619 {} key[fortran] := 'fortran';
620 {} key[forward] := 'forward';
621 key[func] := 'function';
622 key[goto] := 'goto';
623 key[label] := 'label';
624 key[proc] := 'procedure';
625 key[program] := 'program';
626 key[var] := 'var';
627 badlist := nil;
628 if not eof(except) then
629     readbadnames;
630     soan;
631     readinglabels := false;
632     if sy = program
633     then
634         begin
635             write(inter, 'program');
636             complmodule(nil)
637         end (IF)
638     else
639         writeln(' ""program" expected. ');
640
641 13:
642 and ( AUGMENT ).
643 1 *
644 2 *     ANALYZE - ANALYZE AND SUMMARIZE EXECUTION TIME
645 3 *     PERFORMANCE MEASUREMENTS FROM AN AUGMENTED
646 4 *     PASCAL PROGRAM.
647 5 *
648 6 *     S. MATWIN AND
649 7 *     M. MISSALA 1975.
650 8 *     POLISH ACADEMY OF SCIENCES COMPUTER CENTRE.
651 9 *     PKIN, WARSAW POLAND.
652 10 *
653 11 *     MODIFIED, GENERALIZED, AND RENAMED
654 12 *     FROM "PRINRES" TO "ANALYZE" BY:
655 13 *     A. B. MICKEL 77/11/18.
656 14 *     H. U. RUBENSTEIN 77/05/15.
657 15 *     UNIVERSITY OF MINNESOTA COMPUTER CENTER
658 16 *     MINNEAPOLIS, MN 55455 USA.
659 17 *
660     THE NAMES AND ORGANIZATIONS GIVEN HERE MUST NOT BE

```

```

18 *     DELETED IN ANY USE OF THIS PROGRAM.
19 *
20 *     SEE THE PTOOLS WRITEUP (UNDER MEASURE) FOR
21 *     EXTERNAL DOCUMENTATION.
22 *
23 *
24 *     ANALYZE (INTERNAL DOCUMENTATION).
25 *
26 *     ANALYZE READS TWO FILES. INTER2 IS THE FILE CONTAINING
27 *     THE MODULE (PROCEDURE AND FUNCTION) NAMES WHICH ARE USED
28 *     WHEN THE RESULTS ARE SORTED AND WRITTEN OUT. TIMING IS
29 *     THE FILE CONTAINING THE EXECUTION TRACE OF THE PROGRAM
30 *     BEING MONITORED.
31 *
32 *     WITHIN ANALYZE, THE PROCEDURE NAMED PROCESSBODY DOES THE
33 *     ACTUAL ANALYSIS BY DETERMINING EVERY TIME INTERVAL:
34 *
35 *         TIME[EXIT] - TIME[ENTRY].
36 *
37 *     EVERY GOTOEXIT FROM A PROCEDURE IS CONSIDERED TO BE A
38 *     SPECIAL KIND OF PROCEDURE ENTRY, SO THAT ALL PROCEDURES
39 *     WHICH, UP TO THAT TIME HAVE BEEN ENTERED BUT NOT NORMALLY
40 *     EXITED, ARE ALL EXITED BY THE GOTOENTRY. SEE THE COMPANION
41 *     PROGRAM CALLED AUGMENT.
42 )
43
44
45 ($R-,T-,P-,U+)
46
47
48 program analyze(output, inter2, timing);
49
50 label
51     13;
52
53 const
54     alfaleng = 10;
55     maxnames = 2000;
56
57 type
58     alfa = packed array [1 .. alfaleng] of char;
59     tagrange = 1 .. maxnames;
60     measurement = packed record
61         tag: tagrange;
62         time: 0 .. 99999999;
63         mark: (entry, exit, gotoentry)
64     end;
65     counter = record
66         count: integer;
67         name: alfa;
68         timespent: integer
69     end;
70
71 var
72     timing: file of measurement;
73     inter2: file of alfa;
74     modules: array [tagrange] of counter;
75     maxtag,
76     tag: tagrange;
77     proftime: integer;
78     totaltime,
79     totalcalls: integer;
80
81
82 procedure sort(min, max: tagrange);
83

```

```

84 [ QUICKSORT WITH BOUNDED RECURSION DEPTH )
85 ( REQUIRES MIN <= MAX )
86
87   var
88     low,
89     high: integer;
90     midkey: alfa;
91     temp: counter;
92
93   begin
94     repeat {PICK SPLIT POINT}
95       midkey := modules[(min + max) div 2].name;
96       low := min;
97       high := max;
98       repeat {PARTITION}
99         while modules[low].name < midkey do
100           low := low + 1;
101         while modules[high].name > midkey do
102           high := high - 1;
103         if low <= high then
104           begin
105             temp := modules[low];
106             modules[low] := modules[high];
107             modules[high] := temp;
108             low := low + 1;
109             high := high - 1;
110           end;
111         until low > high;
112
113         {RECURSIVELY SORT SHORTER SUB-SEGMENT}
114         if high - min < max - low
115         then
116           begin
117             if min < high then
118               sort(min, high);
119             min := low;
120           end
121         else
122           begin
123             if low < max then
124               sort(low, max);
125             max := high;
126           end
127         until max <= min
128       end {SORT};
129
130
131 procedure processbody;
132
133 ( PROCESS TIMING FILE OF DYNAMIC MEASUREMENTS. )
134
135   var
136     moduletag: tagrange;
137     modultime: integer;
138
139   begin
140     moduletag := timing^.tag;
141     modultime := - timing^.time;
142     set(timing);
143     while timing^.mark = entry do
144       modultime := modultime + timing^.time;
145       processbody;
146       modultime := modultime - timing^.time;
147       if (timing^.mark = gotoentry) <= (timing^.tag = moduletag)
148       then
149         goto 1 ONLY ADVANCE THE TIMING FILE IF A GOTOENTRY

```

```

150
151     IS NOT ENCOUNTERED OR IF A GOTOENTRY IS
152     ENCOUNTERED INTO THE CURRENT MODULE. )
153     get(timing)
154   end;
155   modultime := modultime + timing^.time;
156   totalcalls := totalcalls + 1;
157   with modules[moduletag] do
158     begin
159       count := count + 1;
160       timespent := timespent + modultime
161     end
162   end {PROCESSBODY};
163
164 begin {MAIN PROGRAM}
165   reset(inter2);
166   tag := 1;
167   while not eof(inter2) do
168     begin
169       with modules[tag] do
170         begin
171           read(inter2, name);
172           count := 0;
173           timespent := 0;
174         end;
175         tag := tag + 1
176       end;
177     maxtag := tag - 1;
178     reset(timing);
179     if eof(timing) then
180       begin
181         writeln(' timing file empty. ');
182         goto 13
183       end;
184     proftime := timing^.time;
185     totalcalls := 0;
186
187     processbody;
188
189     totaltime := timing^.time - proftime;
190     page(output);
191     writeln;
192     writeln;
193     writeln(' performance measurement summary for pascal program: ',
194             modules[1].name, '. ');
195     writeln;
196     writeln(' execution time: 62);
197     writeln(' calls: 27, '(milliseconds)': 35);
198     writeln(' module: 9, 'times': 13, 'percent': 11, 'average': 15,
199             'module': 10, 'percent': 11);
200     writeln(' name: 8, 'called': 15, 'of total': 11, 'per call': 15,
201             'total': 8, 'of total': 13);
202     writeln(' -----', '-----': 12, '-----': 11,
203             '-----': 15, '-----': 9, '-----': 12);
204     if maxtag > 1 then
205       sort(1, maxtag);
206     for tag := 1 to maxtag do
207       with modules[tag] do
208         begin
209           write(name: 11, count: 12,
210                 ((count * 100) / totalcalls): 11:3);
211           if count = 0
212             then
213               write('----': 15)
214             else
215               write((timespent / count): 15:2);

```

```

216         writeln(timespent: 9,
217               ((timespent * 100) / totaltime): 12:3)
218     end;
219     writeln('=====: 12, '=====: 11,
220           '=====: 15, '=====: 9, '=====: 12);
221     writeln('totals: 9, totalcalls: 14, '100.000': 11,
222           (totaltime / totalcalls): 15:2, totaltime: 9, '100.000': 12);
223 13:
224 end {ANALYZE}.

```

PROGRAMS

P - 1 Printme

(* Nearly every programming language has to have a program which can reproduce its source text as output. In 1976, Pascal enthusiast, John Strait of the University of Minnesota, (who incidentally maintains the CDC-6000 compiler for the world), wrote such a program, and it is presented below. When Urs Ammann, of E.T.H., Zurich, (who incidentally authored the CDC-6000 compiler), saw this program, he said he had written a shorter one using a case statement. We have not seen it, but we would like to. *)

```

PROGRAM PRINTME(OUTPUT);
(*JPS 76/05/26.*)
CONST FIRSTHALF = 9;
    SECONDDHALF = 10;
    LENGTH = 22;
    Q = ' ';
VAR I,J: INTEGER;
    IMAGE: ARRAY[0..LENGTH] OF
        PACKED ARRAY[1..40] OF CHAR;
BEGIN (* PRINTME *)
IMAGE[ 0] := 'PROGRAM PRINTME(OUTPUT);';
IMAGE[ 1] := '(*JPS 76/05/26.*)';
IMAGE[ 2] := 'CONST FIRSTHALF = 9;';
IMAGE[ 3] := '    SECONDDHALF = 10;';
IMAGE[ 4] := '    LENGTH = 22;';
IMAGE[ 5] := '    Q = ' ';';
IMAGE[ 6] := 'VAR I,J: INTEGER;';
IMAGE[ 7] := '    IMAGE: ARRAY[0..LENGTH] OF';
IMAGE[ 8] := '        PACKED ARRAY[1..40] OF CHAR;';
IMAGE[ 9] := 'BEGIN (* PRINTME *)';
IMAGE[10] := 'FOR I := 0 TO FIRSTHALF DO';
IMAGE[11] := '    WRITELN(IMAGE[I]);';
IMAGE[12] := 'FOR I := 0 TO LENGTH DO';
IMAGE[13] := '    BEGIN WRITE(''IMAGE['',I:2,''] := ' ',Q);';
IMAGE[14] := '        FOR J := 1 TO 40 DO';
IMAGE[15] := '            IF IMAGE[I][J] = Q';
IMAGE[16] := '                THEN WRITE(Q,Q)';
IMAGE[17] := '            ELSE WRITE(IMAGE[I][J]);';
IMAGE[18] := '        WRITELN(Q,' ');';
IMAGE[19] := '        END;';
IMAGE[20] := 'FOR I := SECONDDHALF TO LENGTH DO';
IMAGE[21] := '    WRITELN(IMAGE[I]);';
IMAGE[22] := 'END (*PRINTME*)';
FOR I := 0 TO FIRSTHALF DO
    WRITELN(IMAGE[I]);
FOR I := 0 TO LENGTH DO
    BEGIN WRITE('IMAGE['',I:2,''] := ' ',Q);
        FOR J := 1 TO 40 DO
            IF IMAGE [I][J] = Q
                THEN WRITE(Q,Q)
            ELSE WRITE(IMAGE[I][J]);
        WRITELN(Q,' ');
        END;
FOR I := SECONDDHALF TO LENGTH DO
    WRITELN(IMAGE[I]);
END (*PRINTME*).

```

Articles

Extensions to PASCAL for Separate Compilation

Richard J. LeBlanc
 School of Information and Computer Science
 Georgia Institute of Technology
 Atlanta, Georgia 30332

The lack of features in PASCAL to allow procedures and functions to be compiled separately can be of considerable inconvenience in the development of large programs. This weakness is particularly evident when modifications are being made only to limited parts of a program. Modifications of this sort are common, for example, in the maintenance or extension of a PASCAL compiler.

The extensions described below allow the creation of a global environment, separate compilation of routines using that environment, and additions to the environment without requiring recompilation of existing routines and declarations. Four kinds of modules are recognized to implement these features:

- 1) Declaration modules are used to create an environment.
- 2) Routine modules are used to provide the bodies of routines declared in an environment.
- 3) Environment extension modules are used to make extensions to an environment.
- 4) Main program modules are used to compile the main program body.

Declaration modules

```

<declaration module> ::=
    <<claration heading> <declaration block> .
<declaration heading> ::=
    declarations (<file identifier>);
<declaration block> ::= <constant definition part>
    <type definition part> <variable declaration part>
    <routine heading list>
<routine heading list> ::= <empty> |
    <routine heading> |, <routine heading> |
<routine heading> ::= <procedure heading> |
    <function heading>

```

Within the module may be declarations of constants, types and variables just as in a standard main program. Following these declarations come the headings of routines that are to be part of the environment. These headings are identical to the headings in normal routine declarations.

Any identifier defined in a declaration module may be referenced in any other module compiled using the environment created from the declarations. This mechanism allows routines compiled separately to call each other and to use the same global constants, types and variables. Compilation of a declaration module creates

a description of an environment. This description is used to do all type checking at compile-time, just as if separately compiled routines had been compiled as a standard program.

Routine modules

```
<routine module> ::=
  <routine module head> <routine module body> .
<routine module head> ::= <environment head> routines
  ( <identifier> {, <identifier>} );
<environment head> ::= environment (<file identifier>);
<routine module block> ::= <constant definition part>
  <type definition part> <variable declaration part>
  <routine declaration part>
```

The file in the environment head specifies the environment in which the module is to be compiled. The list of identifiers in the heading tells the compiler which of the routines defined in the module are to match declarations in the environment and thus are to be callable from outside of the module. Routine modules may also contain declarations of constants, type, variables and local routines. The variables so declared are statically allocated and thus retain their values between calls to the routines in the module. This makes routine modules useful for limiting access to data structures to only those routines that need to manipulate and reference them.

A routine module defines a new name scope, so identifiers used in the global environment may be redefined within a module. When a routine declared in the global environment is defined in a routine module, the declaration of its parameters is repeated and the types must match those specified in the environment declaration. (Since the parameter names are not relevant to type checking, they need not match those in the declaration.)

Environment extension modules

```
<environment extension module> ::=
  <extend heading> <declaration block> .
<extend heading> ::=
  extend (<file identifier>, <new file identifier>);
```

Environment extension modules may add any kind of declaration to the environment, but cannot change any existing ones. The environment description from the old environment file is expanded to describe the extended environment and is written as the new environment file.

Main program modules

```
<main module> ::=
  <environment head> <program heading> <block> .
```

Main program modules look exactly like standard PASCAL programs except that the heading is prefixed by an environment heading to supply an environment file

specification. If any routine modules have been compiled in environments produced by extending earlier environment declarations, the main program module must be compiled in the last of the extended environments. Only a linear succession of environments may be used to compile the modules that make up a program.

Experience using separate compilation

These extensions have been implemented in the UW-PASCAL compiler [1,2,3] developed at the University of Wisconsin - Madison for Univac 1100 series machines. Experience using the extensions for further development of that compiler has shown them to be of considerable utility and to provide significant economic advantages. In particular, having the separate compilation features has made it possible to modify and test the compiler within a short time, even during periods of very heavy demand on system resources. Previously, recompilation of the compiler was practical only during off-peak hours.

The UW-PASCAL compiler has also been used by students in a compiler writing course, who made considerable use of the separate compilation features. These students found these extensions to be among the most useful aspects of the compiler. However, no significant reduction in total computing costs was observed in comparison to previous experience using ALGOL and SIMULA 67 compilers. The cost of keeping environment files and relocatable code generated by compilation of routine modules apparently offset the savings in compilation costs. The students' compilers were about 2000-3000 lines long or about 20% of the length of the UW-PASCAL compiler. Some work may be done to determine the program size at which separate compilation provides definite economic advantages in addition to its contributions to convenience and modularity.

References

- [1] UW-PASCAL Reference Manual, Madison Academic Computing Center, 1977.
- [2] Charles N. Fischer and Richard J. LeBlanc, "A Diagnostic Compiler for the Programming Language PASCAL", USE Fall Conference Technical Papers, Lake Buena Vista Florida, October 1976.
- [3] ---, "Efficient Implementation and Optimization of Run-time Checking in PASCAL", SIGPLAN Notices 12, 3, March 1977.

This paper describes work supported by the Madison Academic Computing Center of the University of Wisconsin - Madison.

(* Received 78/02/03 *)

What Are Pascal's Design Goals?

Robert D. Vavra
March 13, 1978

As a long-time reader of Pascal News (PN), I have enjoyed the many articles in which people have discussed various features which could be added to Pascal, but I have been unable to take much of the discussion seriously. In arguing for or against some particular feature, writers have rarely invoked Pascal's design goals in support of their arguments. Such failure to build a proper foundation for one's arguments might be acceptable in casual conversation, but not in a serious discussion.

If a discussion about a language feature is to be taken seriously (by me, at least), the writer must demonstrate that it is firmly based on Pascal's design goals. It is not enough to support a proposed feature by saying that it is easy to use or implement, nor to reject a proposed feature by saying that it is only a "favourite feature". A writer should weigh a proposed feature against each of Pascal's design goals by pointing out which goals favor it and which do not, and should discuss why the tradeoff is desirable.

All of this presupposes that Pascal's design goals are well-understood and generally accepted. In fact, Pascal's design goals are rarely mentioned in PN, so I suspect that they are not well-understood. Further, I think that much of the debate over various language features is really a debate over what Pascal's design goals should be. This article attempts to remedy this situation by summarizing what Wirth's design goals for Pascal originally were, and by starting a discussion of what Pascal's design goals should now be.

In both the original and revised reports [1,2], Wirth's stated design goals are suitably modest:

- * To make available a language suitable to teach programming as a systematic discipline.
- * To allow development of implementations which are both reliable and efficient.

In [3], Wirth stated the following design goals:

- * To make available a notation in which the fundamental concepts and structures of programming are expressible in a systematic, precise, and appropriate way.
- * To make available a notation which takes into account the various new insights concerning systematic methods of program development.
- * To demonstrate that a language with a rich set of flexible data and program structuring facilities can be implemented by an efficient and moderately sized compiler.
- * To demonstrate that the use of a machine-independent language with flexible data and program structures for the description of a compiler leads to an increase of its readability, verifiability and consequently its reliability, and that this gain need not be offset by any loss in efficiency.
- * To gain more insight into the methods of organizing large programs and managing software projects.
- * To obtain a home-made tool which can easily be adapted to other needs.

In [4], Wirth stated the following design goals:

- * To permit clarity and rigour of description by using a small number of fundamental concepts, thereby making program verification easier.
- * To have a wide range of applicability through proximity to actual computer structure, rather than through a host of features collected from various fields of usage.
- * To promote both compile- and run-time efficiency by omitting features which require multi-pass compilation or elaborate run-time support.
- * To promote reliability and efficiency of compilers by providing a language which is simply and regularly structured, thereby allowing the compilers to be simply and regularly structured.
- * To promote machine independence (portability) by extending the definitional capabilities of the language to such a degree of generality that machine dependent entities (types and operations) may appear as special cases selectable by means of predefined names, and whose use presumably enhances the efficiency of programs executed on the particular system in which they are defined.

In [5], Hoare and Wirth summarize all these goals as:

- * To make available a general purpose language efficiently implementable on many computers and sufficiently flexible to be able to serve in many areas of application.

Interested readers can find further details in each of the referenced papers.

As a starting point for a discussion of what Pascal's design goals should now be, I suggest the following list:

General purpose	Pascal should be usable in almost any application area or almost any computer system. There should be reasonably easy ways to manipulate both numeric and non-numeric data, to make use of pre-existing software subsystems (either in libraries or in the operating system), and to implement new general-purpose software subsystems in Pascal.
Introductory	Pascal should be usable by beginning programmers in an introductory programming course. It should be possible for them to write simple programs without needing detailed knowledge of large portions of the language, e.g. I/O.
Low level	Pascal's features should be close to those supported by current computer systems. Any higher-level abstractions (e.g. lists, strings, i/o) should be supported by writing new types and procedures in Pascal.

PASCAL ENVIRONMENT INTERFACE

T. Noodt
University of Oslo

Portable	Software written in Pascal should be reasonably easy to move from one computer system to another. It should be easy for programmers to isolate implementation dependencies in a few places within programs. There should be a minimal number of situations in which the actions of a program are "undefined" or "defined by the implementation".
Small	Pascal should include a minimal number of fundamental concepts.
Systematic	Pascal should encourage the programmer think systematically by allowing him or her to concentrate on a small section of the program at a time. Pascal should minimize the number of special rules which must be learned, instead relying on general rules which apply in all situations.

Obviously, much more needs to be said about Pascal's design goals. For starters, important design goals may need to be added (e.g. reliability, replacement for fortran). Additionally, each of the design goals needs to be more fully explained (e.g. what does "thinking systematically" really mean). Finally, the implications of the entire set of design goals need to be explored (e.g. are the present extension mechanisms powerful enough to allow the language to be general purpose, low level, and small?).

I hope to find the time to write more about Pascal's design goals, and I encourage others (especially those who are proposing language features) to do the same. I look forward to a continuing dialogue in PH on this important topic.

References

1. N. Wirth, The Programming Language Pascal. Acta Informatica 1 (1971).
2. K. Jensen and N. Wirth, PASCAL User Manual and Report. Springer-Verlag (1974).
3. N. Wirth, The Design of a PASCAL Compiler. Software - Practice and Experience vol. 1 (1971).
4. N. Wirth, The Programming Language Pascal and its Design Criteria. Infotech State of the Art Report No. 7, High Level Languages (1972).
5. C. Hoare and N. Wirth, An Axiomatic Definition of the Programming Language PASCAL, Acta Informatica 2 (1973).

(* Received 78/03/20 *)

Opening comments

I am at present working on a Pascal implementation for the Nord 10, running an interactively oriented operating system. (The Nord is a 16-bit Norwegian-made computer. It comes in two variants with 48-bit and 32-bit reals, respectively.) The Pascal Report does not say too much about how to interface a compiler to a computer system and its users. To further complicate matters, what it does say about this relates to a batch system, and is worthless or unusable in an interactive system.

I think that the design of the Pascal environment is fairly important, and that a certain unification would be of value. Below I have sketched a few thoughts about this, and also make some proposals.

why bother

A language is often judged on the way a particular implementation interfaces to its environment, i.e.

- 1) what tools are available to a user for the construction, compilation, and execution of a program, and
- 2) what are the interfaces between the implementation and other systems on the computer (particularly the operating system).

Examples of such interfaces are: - What the available options are, and in what way they can be set or reset. - How a specific file is associated with a file name within a program.

Pascal implementors and fans have chosen to step off the FORTRAN, BASIC, and PL/I highways to enjoy the much nicer view from the Pascal path. The implementors being such rugged individualists, there probably are as many different Pascal environment interfaces as there are Pascal implementations. Implementors are inventing and re-inventing interface features, giving different names to the same feature, or implementing the same feature in a slightly different way.

Since all the big computer vendors soon will become Pascalers (do you doubt it?), the situation will become worse. In vendor A's Pascal implementation, all the extra-language "nice features" will be totally incompatible with those of vendor B. Goodbye portability.

So what

All Pascal implementations need a minimal environment interface, and often a broader interface is highly desirable. There is no reason why this interface should be a completely new one for every new implementation. A little effort can give a lot towards unification, at least in future implementations.

I would like to see a discussion about what such an interface ought to contain, ending up with someone making a recommendation list of features which seem necessary or highly desirable, with the heading:

If you want to include a feature from the list below in your Pascal implementation, it is recommended that you adhere to the specifications stated for that feature.

To initiate a discussion about these matters, I will present my tentative list for the extra-language features in the Pascal implementation I am working on at present.

Options

Option	Effect	Default
L	List program	on
M	List symbolic object code (MAC)	off
Rn	Reals will occupy n words	3
Sn	Sets will occupy n words	8
T	Generate run-time test on indexing, sub-range assignments etc.	on
U	Convert all lower-case letters outside strings to upper case	off

Conditional compilation

The compiler will conditionally skip parts of the source text, depending on the value of flags which can be set by the user. Source lines containing commands to the compiler must have the character \$ in position 1.

flag -> identifier

Command	Effect
\$SET flag	flag gets value <u>true</u>
\$RESET flag	flag gets value <u>false</u>
\$IF flag	Include succeeding source lines if flag is <u>true</u>
\$ELSE flag	End of \$IF of same flag.
	Include succeeding lines if flag is <u>false</u> .
\$END flag	End of \$IF or \$ELSE of same flag.

A flag which has not been assigned a value, will have the value false.

Multiple source files

The compiler command
\$INCLUDE filename
will include the content of that file at this point in the source text. The command may be used recursively.

Command processor

The command processor is a part of the compiler which accepts commands specifying parameters for a compilation.

SET flag

RESET flag

Set and reset conditional compilation flags.

OPTIONS option-list

Set or reset options according to option-list, which has the same syntax as if it appeared within a Pascal comment.

COMPILE sourcefile, listfile, objectfile

listfile and objectfile are optional. The L-option is turned off if listfile is left out.

File access

The procedure OPEN enables association between a specific file and a Pascal file variable at runtime.

```
procedure OPEN(F: filetype; NAME: string;
               var STATUS: integer) . . .
```

F is associated with the file with name NAME. The file is opened, and status for this operation is left in STATUS. The parameters NAME and STATUS are optional. If NAME is not present, the system will enquire the user to specify the file. If STATUS is not present and an error occurs, the job will be aborted if in batch mode or if NAME was specified, otherwise the user will be asked to respecify the file name.

```
procedure CLOSE(F: filetype) . . .
```

The file is closed, and F is disassociated from the file.

Standard procedures

The following standard procedures will be implemented.

procedure TIME(var HOUR, MIN, SEC: integer) . . .

procedure DATE(var YEAR, MONTH, DAY: integer) . . .

function TUSED: real . . .
 (* gives accumulated CPU time in seconds *)

Interactive programs

Several difficult problems arise in the design and running of interactive programs. OPEN and CLOSE take care of run-time association of files.

Another nasty problem is that of reading data from a terminal. If the data does not have the correct syntax, it is of course not acceptable to abort the program. The CERN Pascal implementation has solved this problem in the following manner:

There is a standard procedure
 SETINTERACTIVE
 which when called, will make all error exits from READ save an error status instead of aborting the program. This status can be read with the function COMPLETION.

Closing comments

Some of the specifications above are vague, partly because I do not feel that a long, detailed document is necessary at this stage. Also, your own interpretation or evaluation of a feature may be as good, or better, than mine.

I invite criticism and comments on the features I have described above. You are also invited to add or subtract features.

However, I am not looking for an environment interface list which is as long as possible. Think ecologically, and do not let the environment pollute Pascal!

March 1978

(* Received 78/03/20 *)

SUBRANGES AND CONDITIONAL LOOPS

*Judy M. Bishop
 Computer Science Division
 University of the Witwatersrand
 Johannesburg
 2001 SOUTH AFRICA

The subrange facility in Pascal is an aid to runtime security for fixed boundary constructs such as counting (for) loops and array subscripts. The relevant types can be precisely and naturally defined and the compiler can minimise the amount of runtime checking required. However, an index which increases under program control, as in a conditional (while) loop, presents a problem. This note discusses the problem and presents a solution in terms of a naming convention.

THE PROBLEM

Consider the definitions

```

type index = min .. max;           [D1]
var i : index;
    
```

and the conditional loop

```

i := min;
while (i <= max) and condition do   [D2]
  begin
    (* something *)
    i := succ(i)
  end;
    
```

If the condition remains true, an attempt will eventually be made to set i to succ(max) - a quite normal way of triggering the end of the loop. However, because i's type was precisely defined, succ(max) does not exist! Rewriting the loop with the tests at the end gives a similar error with respect to pred(min), i.e.

```

i := pred(min);                       [D3]
repeat i := succ(i);
  (* something *)
until (i = max) or condition;
    
```

Even without a compiler or program verifier run, it is obvious that these loops are inconsistent with the definitions [D1]. If the loop had only the test on i, then the for statement is the appropriate construct and the undefined nature of the final value is taken care of by the compiler. At least, it should be, but only the B6700 does this. According to Sale's Pascal Compatibility Report, the various final values are

* Previously Judy M. Mullins
 Computer Studies Group
 Southampton University
 ENGLAND.


```

undefined - B6700
max       - CDC6000, Univac 1100
succ(max) - Dec-10, ICL1900, ICL2900

```

Do these options apply equally to succ? Like Sale, I think they should not: an implementation should have a detectable undefined value and succ(max) should yield it. This does not solve the original problem which was to allow a succ(max) for the purposes of controlling a loop.

Typically, this is achieved by weakening the type definition to one of

```

      type index = min .. succmax;           [D4]
or     type index = predmin .. max;
or     type index = predmin .. succmax;

```

This compromise raises its own problem: the subscript type of a corresponding array declaration such as

```
var a : array[index] of item;
```

expects the original index type as defined in [D1] and not one of [D4]'s extended ones.

A SOLUTION

In teaching programming to undergraduate students at Southampton, we made subranges "compulsory". (This can be done by omitting to mention the predefined types integer and real.) We were also blessed with a security-conscious compiler. In order to avoid untold "out of range" errors and general disillusionment in Pascal, we developed the following convention:

1. Subranges are defined over the genuine, natural range of elements, typically that which would be used as an array subscript.
E.g.

```

      type months = 1 .. 12;                 [D5]
      money = minint .. maxint;
      var balances : array[months] of money;

```

2. If an index is required for this subrange, its type is given the same name but prefixed by x- (for "extra") or z- (for "zero").

E.g.

```

      const dec = 12;                         [D6]
      type xindex = min .. succmax
      xmonths = 1 .. 13;
      zmonths = 0 .. dec;

```

3. For an enumerated type, the extra or zero element in the list is named according to the type with the appropriate prefix.

E.g.

```

      type xmonths = (jan, feb, mar, apr, may, jun, [D7]
                     jul, aug, sep, oct, nov, dec, extramonth);
      months = jan .. dec;

```

EXAMPLE

```

function deficit (since : months) : boolean;
var m : xmonths; negative : boolean;
begin
  negative := false; m := since;
  while not negative and (m <= dec) do
  begin
    negative := balances[m] < 0;
    m := succ(m)
  end;
  deficit := negative
end; (* deficit *)

```

Deficit will work with the definitions in [D5] plus either those in [D6] or [D7]. It could be called by

```

      if deficit ((*since the*) 6 (*th month*)) then ...
or by   if deficit ((*in*) dec) then ...

```

This is made possible by keeping max (in this case dec) symbolic and by making use of succ instead of +1. In all compilers known to me, succ(m) is equivalent to m+1, that is, it does not invoke a function call.

COMMENTS

1. Single letter prefixes do, admittedly, hinder readability, but were chosen so as to impinge the least on the length of the original name.
2. A single letter was not considered necessary for the dummy enumerated element since clashes here would be few.
3. The possibility exists for relating a subrange with its extensions in a more formal way. Language designers and pedagogues may care to investigate this.

ACKNOWLEDGEMENTS

Mike Rees, Martin Must and others in the Computer Studies Coffee Room helped convince me that the easy way out i.e. declare one type as the union of all possible variations, should be avoided. The students of Computer Studies II, 1977, proved our solution to be both workable and effective.

The Pascal Compatibility Report can be obtained from

Professor A.H.J. Sale
 Information Science
 University of Tasmania
 Box 252C
 Hobart
 Tasmania 7001.

(* Received 78/02/28 *)

John Nagle
Information Systems Design
3205 Coronado Drive
Santa Clara, CA 95051

(408)-249-8100

Since quite a number of extensions to Pascal have been proposed, I thought that it would be desirable to propose a few deletions to keep the size of the language down. With the goal in mind of keeping Pascal a simple, elegant, and useful language requiring a minimum of run-time machinery, I propose a few simple changes in the direction of simplicity.

1. Get rid of GOTO and LABEL

Need I say more?

2. Get rid of the FORWARD declaration

The FORWARD declaration is a means by which a simple one-pass compiler can find out about calls to procedures not yet defined. Since the compiler can perfectly well figure this out under its own power by capturing the procedure definition at its first call, we should not need to saddle the programmer with this responsibility. At the end of the compilation we can list all undefined procedures. This implementation has the additional advantage that the compiler will not complain so much when we compile programs which are not yet complete, yet will even crosscheck the calls of unwritten procedures for compatibility. This is in keeping with the 'top-down' coding philosophy.

3. Get rid of the 'FOR loop variable problem'

When we get rid of GOTO, we simplify the semantics of the FOR loop by guaranteeing that the index variable of the FOR is always meaningless outside of the loop. Given this simplification, why not simply declare the FOR index variable automatically at the FOR statement AS LOCAL TO THE LOOP. The programmer need not declare it at all, nor should he, since it has no meaning outside the loop. The type of the variable is implicit in the type of the value assigned to it in the FOR statement. This gets rid of the ambiguity associated with the present definition, simplifies

register allocation in the compiler, and is fully compatible with existing correct programs.

4. Get rid of the CASE statement ambiguity

The action of the CASE statement when the CASE selector is out of range is 'undefined'. In one implementation, nothing happens in such cases. In another, the error is always detected and fatal. One of these two actions should be made standard. I would go for making it fatal (and perhaps installing a case name of 'other' to allow the program to capture these cases).

5. Get rid of 'column 1 forms control'

I am aware that this is not a part of the language but of an implementation. However, it is a very undesirable extension as it builds an implementation dependence into programs unnecessarily. The procedure 'page(<file>)' is defined as indicating page ejection. I suggest we use it. Blank lines may be inserted by calls to WRITELN without parameters. If it is desirable to handle the printing of blank lines in some special way to improve printing speed, the WRITE routine should take care of this detail by itself.

6. Get rid of the proposed 'formatted read'

Various parties have noted the very real need for some means of dealing with fixed-format input. Extending READ by adding a formatting facility has been proposed. I see this as an unnecessarily complex approach to the problem. The problem is that of dealing with fixed-format input records. We have a very nice record facility in Pascal; we simply cannot read records easily. I propose that it be permitted to READ any structure of fixed length all of whose base types are CHAR. This form of READ should not be permitted to cross record boundaries (EOLN=TRUE) and the record being read into should be filled with an agreed-upon fill character (probably space, but some persons may want to use an ASCII NUL). This facility also provides the ability to read strings (arrays of CHAR) which is currently an annoying lack. Conversion of numeric fields to values of type INTEGER and REAL can be handled by suitable procedures. It would be desirable to define and standardize such procedures.

John Nagle

(* Received 78/03/09 *)

Open Forum for Members

The University of Tasmania



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

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

Mr. A. Mickel,
Editor, PASCAL News,
Computing Centre,
University of Minnesota,
Minneapolis, Minnesota
U.S.A.

18th January, 1978

Dear Sir,

PASCAL in Australia

May I clarify a few points that have arisen from my letters in #9/10? Firstly the cost of Australian subscription has been criticised for its size (\$A10.00) against the US fee (\$US4.00). Our original budgeting was based on the following quoted figures:

Cost/copy (based on issue #8)	\$2.80	
Postage/copy (in Australia)	.70	(higher in NZ, Malaysia)
	3.50	

I suspected the cost/copy to be high, and we struck a rate of \$2.50/copy as reasonable, giving a subscription of \$10 (4 copies per year). This makes no allowance for othermail (reminders), the necessity to overprint to fill back-orders and possible surplus stock, and higher postage to overseas countries. It must be remembered that the print run in Australia is small, and the postage rates are simply fiendish.

Subsequently, issues 9/10 have been coalesced, and by the time next year comes we shall know better what it really costs. I hope it is less. Preliminary revisions of the costs have shown cost/copy at about \$1.00 and postage in Australia has dropped to 60c, making a fee of \$7.00 possible. I'll give you a figure for 1978/9 after we have processed #11.

The second point I have been taken to task over is a statement of mine that our first-year course switch to PASCAL was "a first for reactionary Australia". Since this seems to have been misunderstood, PASCAL News readers may be interested to know that PASCAL has been in use in CDC universities for some time, notably the Universities of Adelaide, Sydney, New South Wales (and Melbourne). In some cases as an introductory language, in others as a later language.

However, in none of these, nor in any of the other universities that I know, has the combination of a full undergraduate first-year course combined with the use of PASCAL as a first language. However, I don't know everything that goes on in all of the 20 universities spread over the continent, so I apologize if I'm wrong. My clear impression is that FORTRAN still dominates the Australian scene, with COBOL and BASIC hovering around as well.

Yours sincerely,

Arthur Sale,
Professor of Information Science.

Tema

Società per azioni con sede in Roma
Capitale L. 800.000.000 l. v.
Trib. di Roma Reg. soc. n. 1450/72
CCIAA Roma 374468 Bologna 211702

Direzione e uffici:
40122 Bologna / via Marconi 29/1
telefono 287295 (5 linee)

PASCAL USER'S GROUP
c/o Andy Mickel
UCC: 227 Exp. Engr.
University of Minnesota
Minneapolis, MN 55455
U.S.A.

riferimenti da citare nella risposta

026/77/0343/GS/1s

Bologna 11/11/1977

Dear Mr. Mickel,

I wish to submit a few considerations to the Forum of FUG members and PASCAL implementers:

Prologue:

"The new language PASCAL is my own favourite programming language, but only time will tell if it can fall beneath its academic roots"

"Teaching the fatal disease" - R. Holt - SIGPLAN Notice (May 1973).

I work in a Software Consulting Agency that deals with the following main subjects:

- mathematical models;
- management information systems;
- basic software and process control.

Our applications must work on user's defined environments so we design and code programs for different computers. I emphasize: we design and code, we do not teach programming.

Aim:

We think that PASCAL is a very good language (we use it, as design tool, since 1972, and as coding language since 1975) but we also think that a few PASCAL features ought to be improved and a few ones ought to be introduced, on our opinion, in order to provide a good use in commercial applications or, that is almost the same, in day-to-day programming.

"You Know that 80% of commercial application programming is done in COBOL" and there is not an "other language that is a practical alternative for commercial work" - Call for Paper, First European Conference on pragmatic programming & sensible software - 1978, J.Weinberg - With this letter we wish add ourselves to the chorus of other letters on the same matter that we have read on PASCAL Newsletters.

Improvements:

- 1 - Enumerated scalar types: many commercial applications, and not only, use very much this kind of type, and their subranges, but variables of these types are incommunicable via standard Input/Output and external files. In order to have a clear programming is quite useless their inner representation, it would be better their identifiers. A possible solution is to use their names (with common restrictions) as strings in I/O, this is implementable if we do a pre-processor of the compiling stage that is not very heavy compared with the time and space used by other common compilers.
- 2 - Strings: we like the way of defining a string - packed array [1..n] of char - but, in practical programming, strings must be of varying

length - with 'n' as maximum length - With formatted input, it would be better to read the complete string in the same way as it is possible to write it (also without format). Pack and Unpack are useful procedures to process a character in a string but they are not sufficient to use strings in programs, one must provide the complete set of operators.

We think that a good way of re-design this data type is to introduce string (n) as base data type.

3 - Formatted input: we have read the proposed solutions of the quarrel in Pascal Newsletters, but is not a practical day-to-day solution. On the other without formatted input is difficult to use PASCAL in commercial applications because a blank in a (commercial) card is as important as a character.

4 - Case statement: we emphasize the utility to have:

case expression of
value 1:

:

value n:

otherwise:

in addition to actual <case statement>

5 - Interlanguage communication: the "slogan" must be: "we have to manage the transition between old and new systems". But more than this often it happens that one have to use existing packages (data-base, linear-programming) but not also, many times we have to interface a system that, for many reasons, must not be re-done. So it is very important, if we want to introduce the PASCAL language, to have a construct to manage communication with FORTRAN, COBOL, PL/1.


Epilogue

We think it is very good to have a:

- sparse, simple language;
- general purpose language;
- vehicle for portable software;
- tool for systematic programming;
- etc. "

but also that, if software workers do not use it, it means that we have not proposed a real alternative "to dinosaur languages" and to actual programming style.

Sincerely


(Dr. Giuseppe Selva)

J. E. POURNELLE AND ASSOCIATES

12051 LAUREL TERRACE

STUDIO CITY, CALIFORNIA 91604

(213) 762-2256

2 Feb 1978

Dear Mr. Mickel,

I have located a PRIMER of PASCAL and although I haven't obtained it yet--the bookstore is very slow-- it promises to solve most of my bibliographic problems.

In your package with the back issues of the newsletter you noted that you have a good PASCAL for a Z-80 system with disks.

My system is a Cromemco Z-2 with 48 K RAM and 16 K ROM, iCom disks with FDOS-3; CPM is also available. *Also have TABEL cassette interface.*

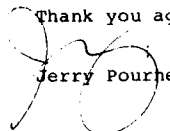
I would be very interested in obtaining a source code for PASCAL that I could get running. I do not expect to get this for nothing, and I am prepared to pay reasonable costs and fees; but I do need to know from whom I can obtain it.

I enclose (1) a stamped self-addressed envelope for your convenience in replying, and (2) a check for \$10 as a donation to the user's group, in the hopes that will provide an incentive for you to take a moment to reply.

As the former President of a writer's association I know very well how volunteer and unpaid work eats up one's professional time and I recall from my grad student days just how little time is available; still, I do hope you'll be able to give me the information on how to acquire PASCAL.

From all I hear, including from friends out at JPL who work in programming the MARINER and PIONEER probes, PASCAL is very powerful and indeed probably better than some of the much-touted and much better known languages. I am about to write a whole mess of software and I would like to have the option of getting that done in PASCAL rather than FORTRAN or any of the 3 BASICS I have.

Thank you again,


Jerry Pournelle

**JOE
CELKO**

Box 11023

Atlanta, GA

30310

USA

Date: 24th Feb 78

To: Pascal News
University Computer
Center
University of
Minnesota
Minneapolis, MN
55455

LEIBNIZ-RECHENZENTRUM
DER BAYERISCHEN AKADEMIE DER WISSENSCHAFTEN
BARER STRASSE 21 D-8000 MÜNCHEN 2

Pascal User's Group
c/o Andy Mickel
UCC: 227 Exp. Engr.
University of Minnesota
Minneapolis, MN 55455
(612) 376-7290

München, den 23.02.1978/HW-1a
Telefax (089) 21 05/84 84
21 05/
Tele: 05/24 634

Gentlemen:

RE: Mr. R. A. Fraley's piece in the last Newsletter on suggested extensions.

I miss dynamic arrays, being an Algol, and I'd love to have real honest string:, too.

However, I don't miss Common, and I don't think we need to have Modules, either.

Common can be dismissed as a horror that allowed each subroutine in FORTRAN to cut up the shared data as it wished (thus Subroutine A was working on four integer, that Subroutine B thought were two Reals, etc.). It was as big a burden as a long parameter list, and it should die with FORTRAN.

There is a nice feature in COBOL, PL/I and some other languages for COPY or INCLUDE verbs that allow the placement of text into the program from other files.

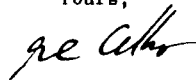
Since this is an operating system-compile time interface we could do the include at the source code level, the intermediate code level, or the binary code level. I favor the idea of doing it as source code, since optimizing work can be done better with constants being plugged in by actual user calls (there was a Student paper in CACM with some timing studies on this technique last year, that would be of interest), and since we must already have a good text file system.

This would mean writing an extension to the compiler you now have that would recognize

(include statement):= include (external file id)
convert it into a comment, and append the proper file(s).

A library would be defined as a procedure or function, a group of procedures or functions, or a group of procedures or functions preceded by a group of declarations. A crippled compiler could check to see that no executable statements were on file. By having the declarations in front of the procedures or functions we would get the shared variables via the existing Global conventions of Pascal, and it would cost us only a few extra lines in a compiler. Portability would not be affected. Optimizing would be possible (for example, deleting items that are not called by the program).

Yours,



Joe Celko

Dear Andy,

enclosed you find a list of wishes of an enthusiastic PASCAL-user. I have condensed it from discussions with some of my colleagues who tried PASCAL for various problems. These wishes are not "filtered" by implementation considerations, but are propositions to make a very attractive language more usable.

As I am preparing a PASCAL course for the users of our computing center, I would be very glad to receive a short note from you indicating the new features of release 3 of the compiler and the approximate date of distribution.

Thank you in advance.

Yours sincerely,



(Hellmut Weber)

Direktorium: o. Prof. Dr. G. Seegmüller (Vorsitzender), o. Prof. Dr. F. L. Bauer, o. Prof. Dr. G. Hämmerlin, o. Prof. Dr. K. Samelson

PASCAL NEWS #12

JUNE, 1978

PAGE 42

A PASCAL-user's viewpoint:

1. Extensions which I consider as necessary
 - 1.1 Dynamic array bounds, for example as describing by WIRTH and further explained by CONDUCT in SIGPLAN letters. (It is ridiculous to have to write a separate inner-product procedure for every vector length. The development of procedure libraries would be drastically simplified.)
 - 1.2 Procedures which are parameters of other procedures must permit var-parameters (You don't have always one single value as result!)
 - 1.3 Formatted i/o as an alternative for the existing READ and WRITE procedures. (It seems to be the only possibility to convince people who write production programs, and until now write them in FORTRAN or COBOL, whether you like it or not.)
 - 1.4 Structured Constants, especially constant arrays (Are needed if one wants to write library routines which are to be called very often by one program, e.g. Series expansions).
2. Extensions which would make programming in PASCAL much more comfortable:
 - 2.1 Break_character
 - 2.2 All characters of a identifier should be significant. (The possibility of suggestive and self-documenting names is severely by the 8-char-rule. Regard
Numberofpossibilities and
Numberofrealizations
and compare it with
Number_of_possibilities and
Number_of_realizations
Even the CDC character.set has still unused characters.)
 - 2.3 Alphanumeric labels (can indicate the condition of their use:
error_in_input_data:)

- 2.4 More standard functions
as y^x , $\log_{10}x$

3. "Extensions" to be included in a sort of "standard" implementation:

- 3.1 Possibilities for interactive use!
- 3.2 Extended file access:
 - 3.2.1 Possibility to append an element to a file after having read the file.
 - 3.2.2 Random access (KNUDSEN (ETH) described a sort of minimal version in PN.)

4. Features every implementation should include:

- 4.1. Simple possibility to generate library routines (e.g. Compiler-option which causes the compiler to assume the dummy main program and to suppress the code generation for it. Non-standard type declarations needed for the library routine had to be placed after the "switching on" of this compiler option.)
- 4.2 Possibility to compile several programs (or a program and separate library routines à la 4.1) in one compiler run.
- 4.3 Debugging aids: (there are no upper limits to the ingenuity of the compiler writer, but there should be at least a dump showing all variables, of structured variables I would accept the bit pattern. See for example the description of H.H. NAGEL in PN#4).
- 4.4 Compiler identification and statistics (Important for documentation and not much work for the implementor.)

5. Especially for CDC-users:

Alternative characters for the important but dangerous characters ':' and '^' .

Release 2 allows '\$' instead of ':' (as proposed by VIM) but I didn't see the fact documented).

This report is a commentary on the contents of issue 11 of News; it may be of interest to PASCALlers in assessing where PASCAL is going and the value of the NEWS.

Page 4: David Barron's proposal for algorithms is excellent! Even if inter-program linkages aren't provided in a PASCAL version, source inclusion always is. I offer any help I can give regarding portability (alas, I am no numerical analyst). Can I urge others to help too?

Pages 8-13: possibly news, abstracts, etc. are the most useful part of News that appears regularly. For me anyway.

Pages 33-34: a good summary of a serious 'compatibility' problem. For a long time I have liked SRTCCO (in the author's terms), but this is something a standard must resolve before I make a change. There are problems in SRTCCO in letting adjustable arrays in, aren't there? It would be interesting to know why the change to SRTCC1 was made in later CDC compilers.

Pages 34-40: A contrast between a tale of woe and rosy future. It is good to see such a lot going on. Anyway, sometime soon it may be possible to present all these 'standard' compilers with a large suite of validation programs to exercise the claims a bit. It's being worked on in the UK by Brian Wichmann, and here by me.

Pages 40-41: Suggestions for compilers and my reactions: (1) I suppose ok, the comment idea is best, but you'd better look out if I find anyone allowing nested comments. (2) Yes; but possibly a good separate cross-referencer would do as well. (3) Must have this; we do. (4) I suppose so; it hadn't occurred to me as a problem as I'd rather remove the packed symbols. (5) Not that we have it, but it is obvious that Kempton isn't a teacher of programming! Do whatever you want if you've got your own version about setting such defaults. They're nothing to do with PASCAL. (6) No comment needed. (7) God help us, why? Cannot input forms be identical to output forms and those in the language too? I regard things like 1. or .4 as quite abnormal; most scientists don't write them that way anyway. (8) What's wrong with Wirth's suggestion of writeln? (9) Some compilers do this now. (10) Possibly pre-defining more constants is a bit of a sledgehammer; a prologue to portable programs can invite users to change a const part of suit their machine.

Pages 41-48: The set of Fraley extensions is too large to comment on. Most of them fall far, far beyond any current ideas of standardization of PASCAL which must include its warts as well as its beauty. I'll only comment on one semantic ambiguity at the end. (1) This is a violation of the principle of environmental independence of procedures; probably an oversight in PASCAL-P. The ICL and Tasmania B6700 do it properly; the compiler is marginally simpler!

Pages 48-53: What a welter of desire to change PASCAL! Can I re-iterate what Andy said: basic PASCAL is not up for grabs so that everyone can add their favourite feature. The Revised Report needs tidying up, yes, after all it was written for communication not as a standard. (Writing a standard calls for the same intellectual effort as proving programs correct.) But not wholesale revision. Only a very few 'agreed extensions' seem worthy of writing down; one that appeared in these pages was allowing of a wider class of type-changing procedures. This might be worth it if it stops people misusing variant records for the purpose (a very difficult thing to detect).

Pages 66-80: Standards. It seems to me now that we could tidy up the loose ends if only all these efforts don't get in each other's way. It'd be a disaster if everyone started modifying compilers to fit into different views. I'm adopting the policy of still fixing bugs and things the current Report is quite clear on, but waiting a bit on any extensions until I see something all are agreed on. It is only a pity this effort is two years or so too late to forestall the variety. I'm delighted to see the bugs in PASCAL-P being brought out; I find I can detect the ancestry of compilers by the results they give to some test programs. And of course errors shouldn't be propagated.

Pages 83-103: Implementation notes: usually not comment-worthy. I feel I must say something about the lousy decisions people are making in the lexical area. Why on earth substitute # for \leftrightarrow , or abbreviate procedure to proc? It is just pointless. Alternately, if the compiler is meant to be used in-house alone, why publicise it in News. Still, enough of gripes. I applaud the setting up of the ICL clearing house. PASCAL now has a canonic implementation for CDC Cybers, Decsystem 10s, UNIVAC 110s, and ICL gear. The IBM and PDP-11 situations are the most worrying, though I think the rumors I get suggest that the AAEC version (see p94) of IBM is quite good (if you keep away from the extensions).

Overview: If this sounded critical in places, please take in it context. I think News is growing up, and the standard of news and contribution is getting much better. I'm glad we are going to have an 'Algorithms Section', and I hope we'll have critical contributions on what's published there. A lot of the portable software I send off for isn't, and requires work to repair the defects which should have been done by the designer. On extensions, I'd really suggest that everyone stop writing them into News since we have too much already. Checklists of troublespots would be welcome, but no new ideas please. For my piece of mind alone, if not Andy's.

Thanks

It is now two years since Andy took up the task of News editor, and just so he doesn't despair and think the task thankless, I'd like to express all PASCALers thanks to you Andy, for the tremendous job you've done. I don't know how you find the time, really I don't. Anyway, thanks.

Arthur Sale



THE UNIVERSITY OF KANSAS / LAWRENCE, KANSAS 666045

Department of Computer Science
18 Strong Hall
913 864-4482

27 February 1978

Pascal User's Group
c/o Andy Mickel
University Computer Center: 227 EX
208 SE Union Street
University of Minnesota
Minneapolis, MN 55455

Dear Andy,

Shame on you! I expect that the only reason you included R. A. Fraley's papers on "Suggested Extensions to Pascal," in PN #11, was to scare the living begabbers out of us. Congratulations. You were terrifically successful. By the end of the second paper I was quaking in my chair (actually, I was expecting to see such things as proposals for long and short int, packed decimal, on condition constructs, sort, and all of the rest of Fortran (ugh!), COBOL (ugh! ugh!), and PL/I (ugh! ugh! ugh!) to be included. Tighten your belt and stand by your guns, I'm sure there's more to come.

Sincerely,

Greg

Gregory F. Wetzel
Research Assistant



eric small & associates, inc.
 consultants in broadcast technology
 680 beach street, suite 365, san francisco, california 94109 telephone (415) 441 0666

March 6, 1978

Pascal User's Group
 C/O Andy Mickel
 University Computer Center:
 227EX
 208 SE Union St.
 Univ. of Minn.
 Minn, MN. 55455

Dear Andy,

We are looking for a PASCAL programmer to join our merry band.

The ideal candidate will be a recent graduate. He or she will have had a lot of PASCAL experience. Exposure to broadcasting operations or engineering would be very desirable, possibly college radio.

Seven people comprise ESA. Dave Rowland, one of the implementors of ESA pascal, is our lead software engineer. We are developing a line of LSI-11 based products for the radio and TV broadcast market.

Sincerely,


 Eric Small
 President

TO: Andy Mickel
 Editor,
 PASCAL NEWS

March 8, 1978

I would like to address certain misconceptions which may have been generated by Professor Arthur Sale's letter in PN#11 (page 75), titled "Unimplementable Features -- Warning".

Professor Sale expressed concern about extensions which some PASCAL implementors have added to their implementations. He claims that these extensions are "not implementable on the Burroughs B5700 system and possibly on other computers." Not only is this claim false in the general case (we are, after all, dealing with computers as powerful as a Turing machine), but there exist relatively simple implementations of two of the three extensions which he uses as examples.

(1) "Passing pointer values as addresses, even in-stack"

Professor Sale's observations in this case are correct, if overstated. It would be inconvenient to implement pointers to non-heap variables on a B6700 system while realizing the advantages of its architecture.

However, there is a more important reason why this extension is not advisable -- the "up-level pointer problem" (and the related "dangling reference problem"). This reason alone, apart from any implementation difficulties, is sufficient to reject such an extension to PASCAL.

(2) "Returning function values of all kinds except files"

Professor Sale has misstated the facts about the B6700 RETN operator. He claims that only operand values (with zero tag) can be returned without causing an "Invalid Operand" interrupt. In fact, however, the B6700 will allow a word with any tag to be returned from a function via the RETN operator. In particular, a data descriptor can be returned quite easily by this method. However, there are other reasons (notably, software conventions) why this approach is not a wise implementation choice.

There is, however, a very clean and simple way of implementing a function return of a descriptor-based type. The most recent release of the B6700 Extended ALGOL compiler (version III.0) includes the new data type "STRING". Values of type STRING can be returned from a typed procedure. The implementation technique for this descriptor-based type may easily be adapted to arrays or records. The B6700 PL/I compiler has used a similar technique for several years, although the new ALGOL STRING implementation is somewhat simpler.

Besides the availability of a suitable implementation, returning such data values from functions seems to be a reasonable language feature. After all, records and arrays may be assigned values and may be passed as parameters by value. There is no fundamental difference between these uses of data values and their use as the returned type of a function.

(3) "Allowing pointers to file-types and the use of new(file)"

While I have argued on grounds independent of implementation that pointers should not be allowed to reference non-heap variables and that records and arrays should be allowed to be returned from a function, I will not take sides on the issue of files in the heap. I will only point out that the restriction that file descriptors must reside in the program stack is a software convention, not a hardware restriction. Furthermore, if the heap is marked as a "dope vector", no more than a few lines of changes to the operating system are required to make files whose descriptor resides in the heap behave normally with respect to being properly closed and deallocated at end-of-task.


 Bob Jardine
 Mission Viejo, California

PASCAL NEWS #12

JUNE, 1978

PAGE 45

March 10, 1978

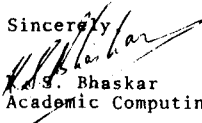
Dear Mr. Mickel:

I maintain PASCAL at the University of Nebraska in Lincoln, and have decided I need my very own personal copy of PASCAL NEWS, so I am enclosing a check for membership for 1977-78.

Looking at the different mutually incompatible versions of PASCAL serves to emphasize one point strongly--that PASCAL needs standardized extensibility. To each his own (idiosyncracies, environments, needs, etc.). If extension mechanisms are developed for PASCAL, as they have for ALGOL 68, a standard PASCAL implementation could be defined as one whose correct programs ran correctly (even if inefficiently) on another standard PASCAL when enclosed by a prelude or environment extension/redefinition block and called as a procedure. Such mechanisms should also enable character set redefinition, reserved word redefinition, etc.

Ideas anyone?

Sincerely,


K. S. Bhaskar
Academic Computing Services

Oslo, March 15, 1978

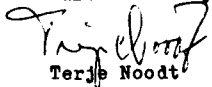
Dear Andy,

I am presently working on a new implementation of Pascal for the Nord-10. The implementation is based on the TRUNC compiler. I would like to communicate with others who have done or plan to do likewise.

Somehow I must interface the compiler to the environment in which it is to be used. This means that I have to invent a set of "features" to go with the implementation - features which already have been invented a lot of times by others.

Trying to turn some of the stones in this field started a train of thoughts, some of which are reflected in the enclosed article. My hope is that the article will provoke a discussion about how the field ought to be plowed.

Yours sincerely,


Terje Noodt

Tektronix
(CORPORATE OFFICE) 15150 NE 28th STREET



Tektronix, Inc.
P. O. Box 500
Beaverton, Oregon 97077

Phone: (503) 644-0161
TWX: 910-467-8708

March 16, 1978

Pascal News
Andy Mickel
University Computer Center: 227Ex
208 S. E. Union Street
University of Minnesota
Minneapolis, MN 55455

Dear Mr. Mickel:

Interest in Pascal has been growing among software engineers at Tektronix for some time. Within Tektronix, wide use is made of many dialects of Pascal for various programming purposes. However, Tektronix does not currently have any products incorporating Pascal programming capabilities. Tektronix will not offer such products unless and until we can do so within our requirements for utility and quality.

Because of the many (somewhat incompatible) dialects of Pascal currently in use and the possibility of Pascal's application to some future products, Tektronix has recently engaged in a study of Pascal extensions. The results of that study are not yet available, but will be made available to the summer workshop proposed by Ken Bowles in Pascal News #11. We expect to offer these results for publication in Pascal News #13.

Sincerely,

TEKTRONIX, INC.



Dr. Don Terwilliger
Manager, Computer Research
Tektronix Laboratories

DT:jlk



RALPH DAVIS
GOVERNOR

State of Florida
Department Of
HIGHWAY SAFETY AND MOTOR VEHICLES
NEIL KIRKMAN BUILDING
TALLAHASSEE 32304

COL. J. ELDRIDGE BEACH, DIRECTOR
DIVISION OF FLORIDA HIGHWAY PATROL
JOHN D. CALVIN, DIRECTOR
DIVISION OF MOTOR VEHICLES
CLAY W. KEITH, DIRECTOR
DIVISION OF PUBLIC SAFETY
AUDRY CARTER, JR., DIRECTOR
DIVISION OF ADMINISTRATIVE SERVICES

March 16, 1978

Mr. Andy Mickel
Editor, Pascal News
University of Minnesota
University Computing Center
227 Experimental Engineering Building
Minneapolis, Minnesota 55455

Dear Andy,

It is such a temptation to detail all my opinions, just because they are mine, even though others have already said the same. Luckily I am pressed for time and the resistance is relatively easy; I think each of the following is in some way new.

1. I enjoy reading Arthur Sale's prolific comments - he is one who often states my opinions. I do object to one aspect of his contributions: his consistent referral to the University of Tasmania's Pascal compiler as "The B6700 compiler". I use a different Pascal compiler on the B6700 (produced by Kenneth Bowles' group at UCSD), know of still another, and hear rumors of one or two others. I expect the Tasmania compiler is a very good one, but it is not the only one.

2. I agree with Arthur Sale's conclusions that certain non-standard features should be avoided. I do not agree with his reasons. These features are not unimplementable on the B6700, as he claims, or the difficulties as horrible as he puts forth. The proper reasons for not implementing these features deal with the language itself, not with a particular implementation. The difficulties encountered on the B6700 are most valuable when used to give insight on future machine design.

3. Formal procedures and functions should be completely specified; the lack thereof is merely a bad holdover from ALGOL60. (I suspect the lack of specification is one reason so many compiler writers omit this feature.) Declaration of procedure types as suggested by George Richmond (PN#8 p 13) leads to such questions as

Are procedure variables allowed?

Should procedures be declared in the VAR section?

Why does a procedure have an initial value (the body) when other variables not?

ad nauseum. These problems should be left to ALGOL68. Therefore the specification should be in-line only. To do so, change the definition of <formal parameter section> to read

Mr. Andy Mickel

-2-

March 16, 1978

```
 ::= <parameter group> |  
    var <parameter group> |  
    <procedure heading> |  
    <function heading>
```

This generates an extra semicolon, so the definitions of procedure and function declaration and heading must be altered to take this into account. This affects pp 112,155 and 159 of the PUM&R; also affected is the program on p 79. Restriction 2 on p 83 can then be dropped. This usage assumes that the type compatibility checking is, in the terminology of Desjardins (PN #11 p 33), SRTCC1; otherwise no two procedures would ever be compatible as types.

4. Standard methods for data transformation are needed, particularly for conversion between character and integer or real; these methods may be functions or procedures or statements. This issue has been much discussed under the guise of formatted I/O. I believe that embedding the transformation of activity into "formatted I/O" unnecessarily complicates the I/O part of the language and unnecessarily restricts the conversion features.

I cannot let Barron & Mullins' argument (PN # p 8) pass unnoticed. Packed data is necessary at times, though formatted I/O is not. My agency handles about 10000 title activity transactions per day, with about 30 fields each.

10000 transactions/day x 30 separators/transactions
= 30000 keystrokes/day

≈ 30 key entry stations&operators

≈ \$30,000/month

= \$360,000/year to use separators.

5. There have been many proposals for extending Pascal's I/O, but usually with no mention of the overall I/O facility which results. Pascal I/O needs improvement, but suggestions should be limited to proposals for a simple, consistent and complete I/O facility, never for isolated features.

Sincerely,

C. EDWARD REID
Kirkman Data Center

CER:jem

Pascal User's Group, c/o Andy Mickel
 University Computer Center: 227 EX
 208 SE Union Street
 University of Minnesota
 Minneapolis, MN 55455 USA

SIEMENS

Three Zeichen und drei Nachstrichstrich

Ungarn Zeichen
 Re

Magyar Október 24-én
 2.1.1978

Dear Andy,

thanks for Pascal News 9 and 10. We finally have completed our work on a version 0 of our portable Pascal-System (cross-version). The first implementation is now on an MDS800 (Intel 8080 microprocessor), using the ISIS II operating system.

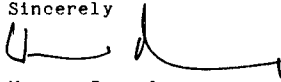
Just some short notes about the history of the project. We decided to implement a portable programming system for the different computers in the Siemens product-field, i.e. 32bit, 16bit and 8bit machines. We came upon Pascal after an implementation of a self-invented language, whose syntax and parts of its data-concept were mainly influenced by Pascal. This language was but experimental and could not be used as base for a programming system.

Our main goals were portability of user-programs as well as portability of the system itself. We think we now have reached both, the first by implementing Standard-Pascal and no dialect at all, the second by using just high-level languages for implementation (mostly Pascal, of course).

Our plans for the near future are a resident version and system-dependent features as a code-generator, generating some form of threaded code. Also a machine-independent dialogue-system has to be developed.

Please do note our new address.

Sincerely



Werner Remmele

Address:
 Siemens AG, Zentralbereich Technik
 Zentrale Forschung und Entwicklung
 Forschungslaboratorien

Bearbeiter
 Remmele

☎ (089) 6782-4622
 Vermittlung 6782 1

☎ 528384

Postfach 837729, D 8000 München 83

SIEMENS AKTIENGESELLSCHAFT
 Zentrale Forschung und Entwicklung - Forschungslaboratorien - Leitung: Prof. Dr. Walter Heywang

Vorstand des Aufsichtsrats: Peter v. Siemens - Vorstand: Bernhard Pfeifer, Vorsitzender - Mitglieder: Theodor Brunner, Dr. J. E. F. von der Planitz, Hans-Joachim Richter, Paul Dax, Max Günther, Hans-Dieter Urich, Hans-Gieter Ködiger, Kurt-Harold Köhler, Friedrich Kehl, Dr. Theodor Zimmmermann - Klaus Hübner, Hans-Joachim Richter, Theodor Brunner, Hans-Dieter Urich, Hans-Gieter Ködiger, Wolfgang Snelig, Hans-Günter Vogesang, Helmut Wilhelm - Sitz der Gesellschaft: Berlin und München - Registered office: Berlin (Inkorporation) 94 188 670; München, 188 6684

Andy Mickel
 Editor, Pascal News

Dear Andy:

Enclosed is my membership form for PUG. You're doing a great job. Keep up the good work!

PASCAL is indeed catching on within Computer Science departments, but, despite the numerous examples mentioned in PN, most other groups I have seen are reluctant to use PASCAL in place of more available and familiar tools. In particular, PASCAL will never replace BASIC or FORTRAN as long as these languages provide features that are sorely lacking in PASCAL. In particular:

FORTRAN has static variables, external compilations, initialization of variables (in particular, arrays), procedures with flexible sized array parameters, STOP and RETURN statements, formatted input with error detection under user control, and large libraries of applications packages. PASCAL does not. In particular, complex numbers would not be missed if a standard subroutine package were available.

BASIC is, despite its lack of power, an extremely friendly language to beginners. Most idiosyncracies are hidden from the user - only one numeric data type, arbitrary length character strings, general FOR loop. Interactive programs are natural. The notion of "One line = one statement" is much easier on the beginner than PASCAL's relatively complicated set of syntax rules. BASIC's editor is very easy to learn.

I have many gripes about PASCAL - mostly concerning features that have been left off. Despite it's goal of being systematic, PASCAL has formatted output (but no formatted input.) It can read and write integers, reals, and characters, (but not enumerated types or records, and to add to the confusion, you can write booleans and packed arrays of characters but not read them!) PASCAL has constants of type integer, real, boolean, character, and set, (but no const declarations of type set, and no record or array constants or constructors.) Functions can return integers, reals, characters, booleans, pointers, enumerated types, and subranges (but not records, sets, or arrays.) There are numerous places where a type identifier is allowed but a type may not be constructed. Semicolons come after most statements (but not before end and never before else.) I also feel strongly that stop, return, exit, and next statements are necessary to promote structured programming.

An else or otherwise clause should be available in case statements. I don't buy Wirth's argument about unstructured programming. He left the goto statement in, so it is quite possible to write poor programs. Give a user enough rope and he can hang himself or climb a cliff. Many members have pointed out the need for a default, so I won't repeat the arguments. Different implementations, unfortunately, use else, otherwise, and $\langle \rangle$ as the default label. I feel the choice should be otherwise. The problem with else is that the preceding statement might be an if - then. The following illustrates the ambiguity:

```

case C of
  'A': writeln('blah');
  'B': if d = e then writeln('blah blah')
      else writeln('error')
end
    
```

The writeln('error') might be an else for the if or for the

case. (I hope implementors who use else have looked at this problem!) The usual kludge is to require a semicolon before the else (how confusing and inconsistent!) but when a user forgets to put the semicolon in, it is possible to get a syntactically correct program, producing an obscure bug that could go undetected for months. (This problem was pointed out by Charlie Fischer.) The case against < as just another label is that it adds nothing to the language: Fraley's example on page 46 of PN #11 is unchanged if you simply remove the semicolon! Of course, using otherwise requires adding another reserved word, but I feel it is the most reasonable solution. The semantics of using an out of range selector with no otherwise should be defined.

Walt Brainerd proposed a loop construction for FORTRAN that solves the "exit in middle of loop" problem (SIGPLAN Dec. 77). Such a construction can be PASCAL-ized and modified to reflect my own biases as is shown in the first syntax diagram. The semantics are that loop ... end cause repetition, and the various other parts give ways to get out of the loop. The for part has the same meaning as PASCAL's for statement: vary the index from the initial value until the final value and then quit. The while part also has the usual semantics:

```
if not <expression> then exit
```

The flag part lists one or more identifiers which should be declared as boolean variables. Entering the loop sets all the variables to false. An exit statement naming a variable sets that variable to true and jumps out of the loop. It is possible to jump out of more than one level of loop by naming a variable in the outer loop's flag part. If no variable is named, none is set, and the innermost loop is exited. The next statement behaves just like the exit except that rather than jumping out of the loop, the remainder of the loop body is skipped and the next execution of the loop begins (after any appropriate incrementing and testing of for and while parts,) and the boolean named is not set to true. (The only purpose of a variable in a next statement is to specify more than one level of loop.) If all three parts are left out of the loop header, an infinite loop results (presumably containing an exit statement somewhere.) Assigning true to one of the flag variables has no effect, is bad style, and might be prohibited.

This construction has a number of advantages. It includes the power of PASCAL's for, while, and repeat statements into one construct. It also has the power of being able to exit or resume one or more levels of loops from any point in the middle. In addition, when you get out of a loop, you can test the boolean variables to see what caused loop termination. Consider for example binary search:

```
l := 1; u := n;
loop while (l <= u) flag found do
  mid := (l+u) div 2;
  if x < A[mid]
    then u := mid-1
    else if x > A[mid]
      then l := mid+1
      else exit found
end;
if found then writeln('Found at',mid)
else writeln('Not found');
```

Another example, finding prime numbers:

```
loop for p := 2 to n flag potential prime do
  loop for d := 2 to trunc(sqrt(p)) do
```

```
    if p mod d = 0 then next potential_prime
  end;
  write(p)
end;
```

The second example above seems to be one case where PASCAL really needs a step option in the for loop, since it is only necessary to check the odd numbers and divisors. What is so all fired important that makes increments other than 1 and -1 against the spirit of PASCAL?

This construction is powerful enough to replace for, while, and repeat loops. A lone for or while part on the loop statement gives you the for and while loops, and a while part on the loop end gives you the repeat ... until construction.

In principle, the while clause is unnecessary, since a conditional exit at the beginning or end of the body will have the same effect. I argue that the while construction provides additional readability. The keyword flag is perhaps not ideal, Brainerd used until, which would only cause confusion in PASCAL. Another keyword, such as conditions, could be substituted.

I'm not suggesting throwing away PASCAL's looping constructions and replacing them with the loop statement. Clearly there is already too much investment in existing programs and compilers, and too little to gain. However, there are several points to learn from. PASCAL's looping constructs, even though far better than what is available in many other languages, still leave much to be desired. Future languages, including a possible PASCAL II, might include it. Alternatively, it might be possible to include the exit and next statements and the flag part in "standardized" extensions. The flag part could be optionally inserted before the do in for and while loops, and after the repeat in that loop. (See the second set of syntax diagrams.)

Work at the University of Wisconsin is currently in progress toward the design and implementation of a PASCAL based Artificial Intelligence language called TELOS. This language is a superset of PASCAL (with two exceptions: goto out of procedures, and passing procedures as parameters, are disallowed) and has numerous extensions (including a clean way to achieve the effect of passing procedure names as parameters.) The language includes features found in many other AI languages, with a special emphasis on the PASCAL philosophy of structured programming, readable code, and detection of errors at compile time.

The language includes capsules (programmer defined data types with their own local operations); coroutines and other synchronous process manipulation facilities; events and handlers for them; an associative data base (referenced with patterns) that can hold objects of any user defined data type, including records, capsules, and the like; different contexts of the data base (so you can make a tentative modification to the data base and see how it compares); multi type pointers; record, array, and pattern constructors; modular compilation; and miscellaneous minor PASCAL extensions (including an otherwise in a case, flexibly sized array types, and functions returning any definable type.) The above loop construction is not part of TELOS, since an effort has been made to avoid cluttering the language up with extra features that can be gotten with existing features, and the event mechanism will provide the same power.

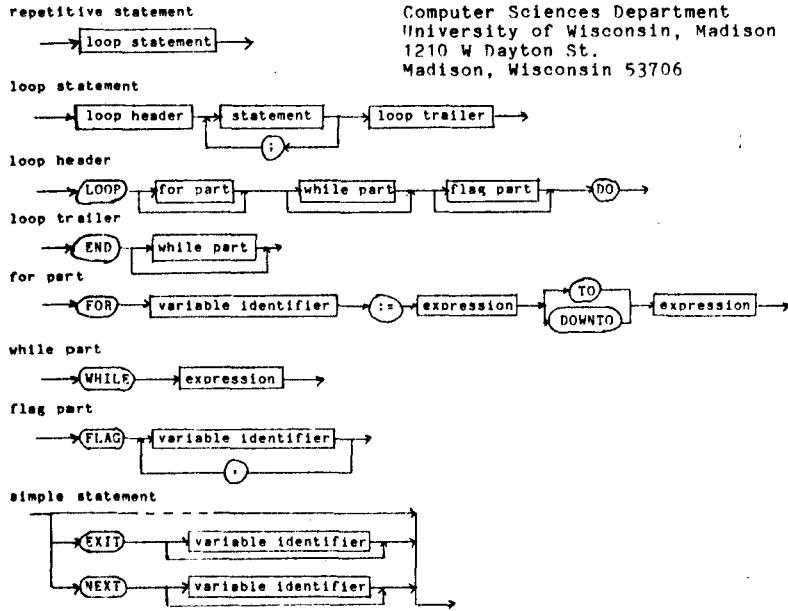
A TELOS implementation is currently under development based on Charlie Fischer's UNIVAC 1100 PASCAL compiler, which currently has most of the data base features implemented. Work is also

beginning on a portable TELOS interpreter based on the P Compiler.

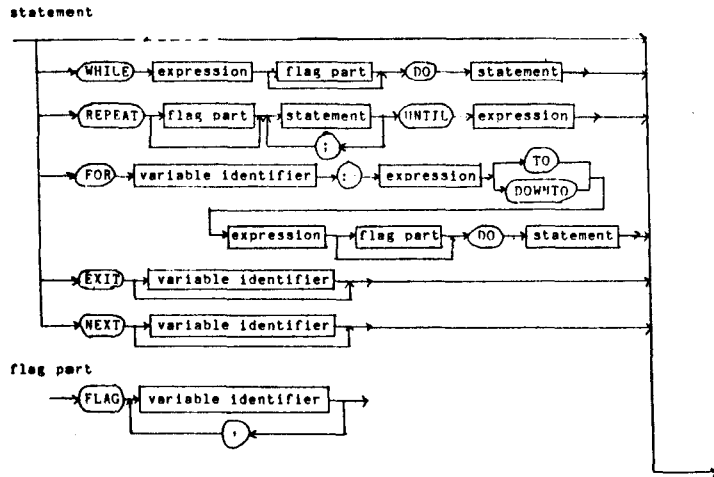
Mark Horton

Mark Horton
Computer Sciences Department
University of Wisconsin, Madison
1210 W Dayton St.
Madison, Wisconsin 53706

Syntax Diagram 1



Syntax Diagram 2



Westinghouse
Electric Corporation

Defense Group



Defense and Electronics Systems Center
Systems Development Division

Baltimore Washington International Airport
Box 746 M.S. 451
Baltimore Maryland 21203

April 11, 1978

Mr. Andy Mickel
Pascal User's Group
University Computer Center 227EX
208 SE Union Street
University of Minnesota
Minneapolis, MN 55455

Dear Mr. Mickel:

I have been aware of PASCAL for several years. Recent interest by Department of Defense in PASCAL as a base for a DoD's Common Programming Language Effort has stimulated my interest. I am deeply involved in the DoD world of software and its unique problems. Possibly PUG can help with one unique problem, i.e. The government requires detailed specifications for everything, including software. Further, the government requires acceptance test to be sure specifications are met. Is it possible for PUG to develop an acceptance test for PASCAL compilers? Don't answer too quickly. An acceptance test, that might satisfy government standards, requires the following:

- 1) a detailed, unambiguous specification.
- 2) A test against every item in the specifications.

As a case in point, Rome Air Development Center has a JOVIAL J73/1 compiler validation (acceptance tests) against MIL-STD-1589, that has over 20,000 source JOVIAL statements in 28 source modules. The JOVIAL validation is compiled and executed. The results of the execution are several thousand "TEST PASSED" or "TEST FAILED" messages with appropriate comments about the language feature being tested.

My group will soon be getting a PASCAL compiler for the UNIVAC 1110. Since the compiler may be used on government contracts, it would be a great help if an acceptance test was available.

Jon S. Squire
Jon S. Squire, Manager
Operational Software

JSS:j

University of the Witwatersrand, Johannesburg

DEPARTMENT OF APPLIED MATHEMATICS

1 Jan Smuts Avenue, Johannesburg, 2001, South Africa
Telephone 39 4011, Telegrams 'University', Telex 8 7330 SA

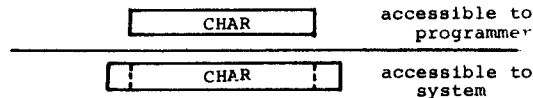


The Editor,
Pascal News.

telephone ext
your reference
our reference JB/SW
date: 7 April 1978

Dear John,

Thanks for your note - the question of predefined types also requiring a succ(max) facility had not occurred to me when I wrote "Subranges and Conditional Loops". The convention I suggest only works for genuine subranges, not full ranges such as integer, char and boolean. I thought long and hard of the possibility of letting these types be subranges of underlying and hidden ranges. These ranges would be one bigger on either side than the subrange we see but these "fringe" elements would not be accessible. Diagrammatically, we want



The idea would be to let a program declare
and write var ch : 0.. succcharmax

```

ch := charmin;
while (ch <= charmax) and condition do
begin
  (* something *)
end;
ch := succ (ch)

```

The trouble is that the fringe elements are accessible: if succ(ch), when ch=charmax, is a valid expression then there is no way of stopping a program from writing it out - which would be invalid. Furthermore, there may be severe implementation problems since these types have a "fully packed" property i.e. they are usually represented in the exact number of bits required for max.

This leads me to realize that the predefined types, namely char, boolean and integer, are ranges and have a different nature to the subranges that we build on top of them. For the first time I feel some sympathy with Haberman and his "Critical Comments":

...../2

- 2 -

To obtain the full effect of the above program in standard Pascal requires a boolean i.e.

```

var ch : char;
ch := charmin;
indexed := false;
while not indexed and condition do
begin
  (* something *)
  if ch = charmax then indexed := true
  else ch := succ (ch)
end;

```

This use of booleans is similar to that required to simulate sequential conjunction. I must admit that I don't like it and wonder if one day we'll have a "Booleans Considered Harmful" article.

I would be very pleased to hear if other Pascal people have thought about this problem and have alternative views to mine.

Enclosed are some membership forms - dollars are coming separately by Postal Money Order.

Best wishes,

Judy
JUDY BISHOP

Encl.

(* Note: This letter is in reply to a letter sent on 78/03/08 from John Strait to Judy:

"Belated congratulations to you and Nigel! We received your card--you two make a handsome couple. Andy let me read your article "Subranges and Conditional Loops" which he received yesterday. I have a question: What do you do with a pre-defined type which cannot be redeclared (e.g. CHAR) or one with special meanings (e.g. BOOLEAN). I ran into this problem last week with CHAR. Aside from this problem, I found your solution interesting/elegant."

*)

PASCAL STANDARDS

PLEASE DIRECT ALL INQUIRIES, LETTERS,
ETC. ABOUT Pascal Standards to
Tony. Thanks, Andy.

Editor: Tony Addyman Department of Computer Science
The University
Manchester M13 9PL
United Kingdom
(phone: 44-61-273-7121 x5546)

Jim Miner and I would like to bring you up to date on recent standards developments. First, Ken Bowles at the University of California, San Diego, has failed to keep us informed about his proposed summer workshop. We have no news since last issue!

Beginning this January, Jørgen Steensgaard-Madsen of the Datalogisk Institut, University of Copenhagen, Denmark, has done all Pascalers a favor by initiating work on the difficult task of conventionalizing extensions--thus answering Pierre Desjardins's good question in the last issue of PUGN. Jørgen is working in cooperation with Niklaus Wirth and several implementors: Jeff Tobias and Gordon Cox (IBM 370) in Australia, H. H. Nagel (DEC 10), in Germany, Olivier Lecarme (CII IRIS) in France, John Strait (CDC 6000) in the USA, Arthur Sale (B6700) in Australia, Ken Bowles (DEC PDP-11 and micros) in the USA, and Jim Welsh (ICL 1900) in the UK.

Olivier Lecarme published letters from Niklaus Wirth in the Bulletin No. 3 for the French Working Group on Pascal in March. The hope was expressed that this is hopefully the final work done in this area and that progress could be made if the number of people were kept small and the range of topics to be considered kept limited.

Jørgen is in contact with Tony Addyman who continues making progress on an ISO standard with his 10- (so far we at PUGN don't know who they all are)-person BSI working group called DPS/13/4. Tony now expects to have a draft document by the end of September.

In the course of our correspondence with Jørgen and Niklaus, we discovered another standardization effort begun by Justin S. Walker at the National Bureau of Standards (NBS) within the U.S. Government. He coincidentally (?) joined PUG 2 weeks later. We sent a personal letter to him trying to determine just what he is doing, and he did not answer. ~~Hummmmm~~

Below is news from Tony and DPS/13/4: an Attention List #2.

Following that are several letters. Charles Fischer of the University of Wisconsin and Richard LeBlanc of Georgia Institute of Technology have stated very clearly some widely-held concerns over standards. Jim and I wholeheartedly agree with them.

Bob Vavra has written an outstanding and timely article and letter on design goals.

Arthur Sale has issued a revised version of his "Pascal Compatibility Report" [Department of Information Science Report No R78-3, May, 1978] which we described last time in this space. It now includes many more implementations.

Arthur is working with Brian Wichmann, of the National Physical Laboratory, in the United Kingdom on a set of Pascal programs to do: 1) Validity Checks - Does the compiler accept standard code, normal, or wierd? 2) Quality Checks: How does the compiler cope with error and error recovery? and 3) Compatibility Checks: How does the compiler cope in the undefined areas?

- Andy and Jim

April 7, 1978

Dear Andy,

I enclose a Pascal syntax written in EBNF. Would it be of any interest to the Newsletter?

Best regards,

Niklaus

Prof. Niklaus Wirth

ETH EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE
ZÜRICH

Institut für Informatik

PASCAL syntax

NW 12.3.78

(Extended BNF: cf. Comm.ACM 20, 11, p. 822, Nov. 1977)

identifier = letter {letter | digit}.
IdentList = identifier {"," identifier}.
UnsignedInteger = digit {digit}.
UnsignedReal = UnsignedInteger ["." digit {digit}] ["E" ScaleFactor].
sign = "+" | "-".
ScaleFactor = [sign] UnsignedInteger.
UnsignedNumber = UnsignedInteger | UnsignedReal.
String = "\"" character {character} "\".

ConstantDefinition = identifier "=" constant.
ConstantIdentifier = identifier.
constant = [sign] (UnsignedNumber | ConstantIdentifier) | string.

TypeDefinition = identifier "=" type.
type = SimpleType | StructuredType | PointerType.
SimpleType = TypeIdentifier | ScalarType | SubrangeType.
TypeIdentifier = identifier.
ScalarType = "(" IdentList ")".
SubrangeType = constant ".." constant.
StructuredType = [PACKED] (ArrayType | RecordType | SetType | FileType).
ArrayType = ARRAY "(" SimpleType {"," SimpleType} ")" OF type.
RecordType = RECORD FieldList END.
FieldList = [FixedPart] [VariantPart].
FixedPart = RecordSection {";" RecordSection}.
RecordSection = [IdentList ":" type].
VariantPart = CASE {identifier ":" TypeIdentifier OF variant {";" variant}.
variant = [CaseLabelList ":" "(" FieldList ")"] .
CaseLabelList = constant {"," constant}.
SetType = SET OF SimpleType.
FileType = FILE OF type.
PointerType = "~" TypeIdentifier.

VariableDeclaration = IdentList ":" type.
variable = identifier {index | "." identifier | "!"}.
index = "[" expression {"," expression} "]" .

expression = SimpleExpression [relation SimpleExpression].
relation = "=" | "<" | "<=" | ">" | ">=" | IN.
SimpleExpression = [{"+" | "-"} term {AddOperator term}].
AddOperator = "+" | "-" | OR.
term = factor {MulOperator factor}.
MulOperator = "*" | "/" | DIV | MOD | AND.
factor = variable | UnsignedConstant | FunctionDesignator | set |
"(" expression ")" | NOT factor.
set = "{" [element {"," element}] "}".
element = expression ["." expression].
FunctionDesignator = identifier [ActualParameterList].
UnsignedConstant = UnsignedNumber | string | ConstantIdentifier | NIL.

statement = [label ":"] UnlabelledStatement.
UnlabelledStatement = SimpleStatement | StructuredStatement.
SimpleStatement = [AssignmentStatement | ProcedureStatement | GotoStatement].
AssignmentStatement = variable "=" expression.
ProcedureStatement = identifier [ActualParameterList].
ActualParameterList = "(" expression {"," expression} ")".
GotoStatement = GOTO label.
label = UnsignedInteger.

```

StructuredStatement = CompoundStatement | ConditionalStatement |
                    RepetitiveStatement | WithStatement.
CompoundStatement = BEGIN statement {";" statement} END.
ConditionalStatement = IfStatement | CaseStatement.
IfStatement = IF expression THEN statement [ELSE statement].
CaseStatement = CASE expression OF case {";" case} END.
case = [CaseLabelList ":" statement].
RepetitiveStatement = WhileStatement | RepeatStatement | ForStatement.
WhileStatement = WHILE expression DO statement.
RepeatStatement = REPEAT statement {";" statement} UNTIL expression.
ForStatement = FOR identifier ":@" ForList DO statement.
ForList = expression (TO | DOWNTO) expression.
WithStatement = WITH variable {";" variable} DO statement.

```

```

ProcedureDeclaration = ProcedureHeading block.
ProcedureHeading = PROCEDURE identifier [FormalParameterList] ";".
FunctionDeclaration = FunctionHeading block.
FunctionHeading = FUNCTION identifier [FormalParameterList] ":"
                TypeIdentifier ";".
FormalParameterList = "(" FormalParameterSection
                    {";" FormalParameterSection} ")".
FormalParameterSection = [VAR | FUNCTION] IdentList ":" TypeIdentifier |
                        PROCEDURE IdentList.

```

```

block = [LabelDeclarationPart] [ConstantDefinitionPart] [TypeDefinitionPart]
        [VariableDeclarationPart] ProcedureAndFunctionDeclarationPart
        StatementPart.
LabelDeclarationPart = LABEL label {";" label} ";".
ConstantDefinitionPart = CONST ConstantDefinition ";" {ConstantDefinition ";"}.
TypeDefinitionPart = TYPE TypeDefinition ";" {TypeDefinition ";"}.
VariableDeclarationPart = VAR VariableDeclaration ";" {VariableDeclaration ";"}.
ProcedureAndFunctionDeclarationPart =
        {ProcedureDeclaration ";" | FunctionDeclaration ";"}.
StatementPart = CompoundStatement.

```

```

program = ProgramHeading block "." .
ProgramHeading = PROGRAM identifier "(" IdentList ")" ";".

```

SYMBOLS

```

+ - * / := . , ; : " = <> <=> >= ( ) [ ] ↑ ..
(* *)

```

Keywords

```

AND ARRAY BEGIN CASE CONST DIV DO DOWNTO ELSE END
FILE FOR FUNCTION GOTO IF IN LABEL MOD NIL NOT
OF OR PACKED PROCEDURE PROGRAM RECORD REPEAT
SET THEN TO TYPE UNTIL VAR WHILE WITH

```

Predeclared identifiers

```

ABS ARCTAN BOOLEAN CHAR CHR COS DISPOSE EOF EOLN
EXP GET INPUT INTEGER LN NEW ODD ORD OUTPUT PRED
PUT READ READLN REAL RESET REWRITE ROUND SIN SQR
SQRT SUCC TEXT TRUNC WRITE WRITELN

```

PROFESSOR OF COMPUTER SCIENCE
 T. KILBURN C.B.E., M.A., Ph.D.,
 D.Sc., F.I.E.E., F.B.C.S., F.R.S.
 ICL PROFESSOR OF COMPUTER ENGINEERING
 D. B. G. EDWARDS, M.Sc., Ph.D., M.I.E.E.
 PROFESSOR OF COMPUTING SCIENCE
 F. H. SUMNER, Ph.D., F.B.C.S.
 PROFESSOR OF COMPUTER PROGRAMMING
 D. MORRIS, Ph.D.



DEPARTMENT OF COMPUTER SCIENCE
 THE UNIVERSITY
 MANCHESTER
 M13 9PL

Please note our **new**
telephone number

061-273-7121

Andy Mickel
 PUG
 etc.

061-273-7121 X5546
 6 February 1978

Dear Andy

This letter will serve several purposes. These are:

- To tell you my new phone number for the roster.
- To give and all others at PUG central the latest Attention List. As I said in my call, don't be alarmed by some of the items I still operating on the same basis as before.
- I want to include several paragraphs from the beginning of PASCAL News as an appendix to the textbook. This will serve to advertise PUG by describing PASCAL News, giving the central and regional addresses etc.. If space permits I would like to include a copy of the all-purpose-coupon. Will this be OK? This does not need a reply. I will call you. If you cannot be around, you can always leave a simple yes/no answer.
- A reminder to all PUG members that any contributions to the standardisation effort will be gratefully received by myself and other members of DPS/13/4.
- As the enclosed material is rather bulky, you may print it or not as you see fit. Hopefully I will be able to send you some more up-to-date information before the next issue is due to be printed.

Yours

Tony

A. A. ADDYMAN

Rec'd 78/2/21

PROFESSOR OF COMPUTER SCIENCE
T. KILBURN, C.B.E., M.A., PH.D.
D.Sc., F.I.E.E., F.B.C.S., F.R.S.
ICL PROFESSOR OF COMPUTER ENGINEERING
D. B. G. EDWARDS, M.Sc., Ph.D., M.I.E.E.
PROFESSOR OF COMPUTING SCIENCE
F. H. SUMNER, Ph.D., F.B.C.S.
PROFESSOR OF COMPUTER PROGRAMMING
D. MORRIS, Ph.D.



DEPARTMENT OF COMPUTER SCIENCE
THE UNIVERSITY
MANCHESTER
M13 9PL
Telephone: 061-273 5466

1st February 1978

TO: Members of DPS/13/4, the Swedish Technical Committee on Pascal and all Correspondants.

May I first apologise for the delay in the production of Attention List #2. I (wrongly) decided to keep altering the list to include new material. Had I realised that it would take such a long time to produce the list, I would have issued an incomplete list earlier.

It is my hope that the list contains all the doubts and problems concerning Pascal which have been brought to my attention. If this is not so, then the list will be updated.

Since it is a long time since DPS/13/4 last held a meeting, and several of its members are very active, I am suggesting that a meeting be held in late February or early March. The meeting cannot be called too soon because the Swedish Technical Committee will need time to arrange for their representative(s) to be present.

Progress is being made in several related areas:

1. BSI is proposing the creation of an ISO project for Pascal.
2. Brian Wichmann is endeavouring to create a suite of Pascal test programs. This is an encouraging move for DPS/13, who collectively believe that validation suites for compilers should be provided wherever possible.
3. Prof. N. Wirth is optimistic concerning our copyright problems. Springer-Verlag have not yet replied to my letter.

To avoid further delaying this attention list, the following items will be sent separately later in the month:

1. A list of names (and where appropriate addresses and telephone numbers) of all people actively involved in the standardization effort.
2. A list of other people who are being kept informed of progress.
3. A selection of the guidelines and rules from BS:0, Part 3, which concerns the way in which the working group should operate.
4. A summary of the relevant parts of "Guidelines for Approving Standards", which is part of a document presented by the United Kingdom to ISO at the Hague meeting of the programming languages sub-committee. Although this was not accepted by ISO, it may be necessary for BSI to adhere to its own requirements.
5. Sections and sub-sections of the report which individual members of the working group are requested to study. Any member of the group is, of course, quite free to study and make suggestions concerning any part of the report.

A.M. Addyman
Convener of DPS/13/4.

March 23, 1978

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

ACADEMIC COMPUTING CENTER
THE UNIVERSITY OF WISCONSIN - MADISON

1210 WEST DAYTON STREET
MADISON, WISCONSIN 53706
608-262-1100

Dear Andy:

As an implementor of a PASCAL compiler as well as a "firm believer" in PASCAL's merit as a programming language, I feel compelled to comment on Ken Bowles' recent proposal (PASCAL News #11) to convene a workshop to standardize PASCAL extensions. The value of standardizing the extensions all implementors (including this one!) seems to add to PASCAL is unquestionable. However, Bowles' approach seems to me to be rather suspect. If this standard is to have any real value, it must have broad-based support in the PASCAL user community.

Why then should the effort to produce this standard be dominated by organizations with a large monetary investment in PASCAL with the gratuitous inclusion of "a small number of academic experts" to placate the rest of us? Are we to believe the opinions of the average PASCAL devotee are less important than those of industrial and governmental organizations? Such an idea strikes me as rather odd given the fact that PASCAL has succeeded not because of these organizations, but in spite of them.

Even if the composition of Bowles' workshop is made more equitable and broad-based, I have very serious reservations about any language designed by committee. It can be very strongly argued that PASCAL's simplicity and elegance derives directly from the fact that it was designed by one man. Why not then adhere to this principle?

Bowles' workshop should by all means meet (although with a more broad-based composition). Rather than drafting a specific set of language extensions, however, it should draft a set requirement that an extended PASCAL should meet. Where necessary, differences in emphasis or opinion should be included--all concerned parties must have a say in what they feel is important. These requirements should then be forwarded to a very small group of acknowledged language design experts (Nicklaus Wirth would, of course, be ideal) who would produce a single set of specific language changes consonant with the "spirit" of PASCAL, the state of the art, and the overall requirements given them. This set of changes would then be widely distributed, discussed and debated, but accepted or rejected as a whole. If they are rejected, (say by vote of the PUG membership) then we must acknowledge that no standardization is, at present, possible. If they are approved, we should accept them as the one and only definition of extended PASCAL.

I realize, of course, that my opinion of how standardization should be done is just one man's viewpoint. However, the principle that everyone should have a voice in what kind of extensions should be included while limiting to a very few experts the decision of exactly how these extensions are to be specified seems a sound one. If we are to produce a standard, let us make every effort to ensure it is something we can live with.

Sincerely,

Charles N. Fischer

April 10, 1978

Mr. Andy Mickel
Editor PASCAL News
University Computer Center: 227 EX
208 SE Union Street
University of Minnesota
Minneapolis, MN 55455

Dear Andy:

I wish to make a few comments about Ken Bowles' proposal (PASCAL News #11) for a workshop to develop a set of standardized PASCAL extensions. As an experienced PASCAL user and an implementor of a PASCAL compiler, I certainly agree that there are areas in which PASCAL could be improved as a systems programming language. There is no doubt that some standardization of extensions would be quite valuable to both users and implementors. However, I have some reservations about the process Bowles has proposed to develop these extensions.

Having had some experience in designing programming languages, I am quite concerned that a set of extensions produced by a large committee might not be consistent with the simplicity that is one of the most attractive characteristics of PASCAL. This simplicity probably results from the fact that PASCAL was designed by one man. It might also be noted that while the DoD "Ironman" project mentioned by Bowles included input from a great number of people in identifying goals, the actual language design work is being done by small groups. I think it more appropriate that any large workshop produce a statement of requirements rather than a "finished" language design.

The fact that attendance at the workshop will be restricted to a certain group of PASCAL users is also of concern to me. If the language to be produced by the workshop would end up being used by only the participants, this would not be objectionable. However, any extended PASCAL standard adopted by a group of users with considerable economic influence is likely to become a de facto standard. It is not acceptable for the PASCAL user community to have so little influence in such an effort. Further, there are apparently other standardization efforts under way. These should certainly not be ignored.

If Bowles wishes his workshop to produce a systems implementation language designed by and for industrial firms and government agencies, the language should be given a name that does not contain "PASCAL" so that there will be no confusion as to its nature. If the workshop is to make a more valuable contribution toward the standardization of PASCAL extensions, a broader group of participants is necessary and more care must be taken to insure that the resulting language reflects the "spirit of PASCAL" and is acceptable to PASCAL users in general.

Sincerely,

Richard J. LeBlanc

Richard J. LeBlanc,
Assistant Professor

Dear Andy,

In your recent article on Pascal Standards (PN 11 page 65), you and Jim point out that many people are suggesting (and implementing) changes and extensions to Pascal, but few are using Pascal's design goals to evaluate their changes or extensions (at least few are doing it publicly). By referring readers to the design goals listed on the back cover of PN, you imply that Pascal's design goals are well-understood and generally accepted. I disagree on both counts.

- * The ten-line description of Pascal's design goals is adequate for the back cover of PN, but it is too vague for use in judging proposed language features. For example, "general purpose but not all-purpose" is a very nice phrase which is intuitively appealing, but it gives little or no guidance to someone who is attempting to evaluate a proposed feature. If I didn't know you, I would be tempted to suggest that you are purposely fostering confusion in this area to keep up the volume of provocative language proposals in PN.
- * You have been exhorting people to justify their proposals in terms of Pascal's design goals since PUG was formed (PN 5 page 2). The fact that few people have done so seems to indicate that many disagree with or do not understand the stated design goals (or else they don't read your editorials).

I agree with you that any future development of Pascal, beginning with the Standardized Extensions that you call for, must be guided by a clearly-defined set of design goals. Discussion of Pascal's design goals has been remarkably absent from most material appearing in PN. I enclose an article which opens such a discussion.

I am pessimistic about our ability to standardize anything beyond the Revised Report (which Audyman seems to have well in hand), without institutionalizing Pascal to a very great degree. (Doing language design by committee is very difficult, but doing it in the pages of PN by a committee-of-the-whole is unthinkable.) The Simula Standards Group (as described by Falme in Software Practice and Experience 1976 pages 405-409) seems to be successful because it has a representative from each of the eight Simula implementations. A Pascal standards committee consisting of a representative from each Pascal implementation would not be a viable design group, and a committee which is any less representative would have difficulty achieving widespread adoption of its standards.

Even so, I am optimistic about the future. It will not be the end of the world (or of Pascal) if we fail to standardize. Pascal's greatest weakness (a multiplicity of incompatible implementations) is a direct result of its greatest strength (simplicity and clarity of programming). The Pascal compilers that I have seen are sparsely documented (to say the least), but many users have been able to modify them to accept new features. Can you imagine as many people doing that to Fortran or Cobol compilers?

Sincerely,

Bob Vavra
Bob Vavra

Implementation Notes

CHECKLIST

Please note the new Checklist entry number 0: DATE/VERSION. In preparation for next year, we encourage implementors, users, or anyone else to submit new or revised checklist reviews for their favorite implementations.

Pascal Implementations Checklist

0. DATE/VERSION
(* Last checklist changes; version name or number, if any. *)
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
(* In form of supplement to Pascal User Manual and Report?
Machine retrievable? *)
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
(* Libraries of subprograms available? Facilities for
external and FORTRAN (or other language) procedures
available? Easily linked? Separate compilation available?
Automatic copy of text from library into source program
available? Symbolic dumps available? *)

PORTABLE PASCALS

Pascal P4 -- Bug Reports

Due to a fit of oversight, I forgot to print in issue #11 Chris Jacobi's Updates 1 and 2 to P4. They appear below. Also, there were a couple of errors in the bug list in issue #11. Bob Fraley caught one (see his letter below). The other error was in bug number (17.), in which the fix should have read:

```
Replace PASC.2826 with:  
THEN ERROR(131);
```

```
Replace PASC.2831 with:  
ERROR(131);
```

- Jim Miner

HEWLETT  PACKARD

3500 Deer Creek Road, Palo Alto, California 94304, Telephone 415 494-1444, TWX 910 373 1267

March 22, 1978

Mr. Jim Miner
25 Blegen Hall
University of Minnesota
West Bank
Minneapolis, Minnesota 55455

Dear Jim,

The fix for set declaration error checking is incorrect. In particular, if any error occurs, LSP is not set and therefore FSP is set to a bad value. To maximize error checking of uses of this type, I would suggest the following fix:

```
Replace PASC 1275 by:  
IF LSP1 = REALPTR THEN  
  BEGIN ERROR(114); LSP1 := NIL END  
ELSE IF LSP1 = INTPTR THEN  
  BEGIN ERROR(169); LSP1 := NIL END  
ELSE  
  BEGIN GETBOUNDS (LSP1, LMIN, LMAX);  
    IF (LMIN < SETLOW) OR (LMAX > SETHIGH) THEN  
      ERROR(169);  
  END;
```

This will build a "SET" type node, checking the use of variables which have this type. (Alternatively, set LSP: = NIL before PASC 1271, and remove "LSP1: = NIL" from line PASC 1273).

The correction to field list, allowing ";" before the "END" of a record definition, is incomplete. In particular, the syntax allows null field entries (multiple ";" in a row). The full fix is:

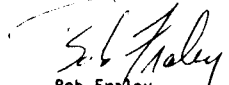
```
Replace PASC 1077 by  
  WHILE SY = SEMICOLON DO  
Change PASC 1079 to:  
  IN FSYS + [IDENT, CASESY, SEMICOLON]  
Change P 150 to  
  IN FSYS + [SEMICOLON]
```

There is another error in P4 which causes an infinite loop when a comment is not closed.

Replace PASC 509 by
UNTIL (CH = ')') OR EOF (INPUT);

Replace PASC 507 by
WHILE (CH <>'*') AND NOT EOF (INPUT) DO NEXTCH;

Sincerely yours,


Bob Fraley
Hewlett-Packard Laboratories
Electronics Research Laboratory

RAF/hma

(* Thanks, Bob! *)

Pascal P4 — UPDATE1 and UPDATE2

Both of these updates are dated January, 1977. They were issued by Chris Jacobi of ETH, Zurich, and we printed them in issue #8.

UPDATE1:

Replace BOOT.4 by:
FOR I := ORDMINCHAR TO ORDMAXCHAR DO SOP[CHR(I)] := NOOP;

Replace P.477 by:
LOAD; GENLABEL(LCIX);

Insert after P.479:
GENUJPKJP(57(*UJP*),LCIX);

Replace P.147 by:
BEGIN ALIGN(LSP1,DISPL);

Replace P.424 by:
LOCPAR := LOCPAR + PTRSIZE;
ALIGN(PARMPTR,LOCPAR);

Insert after line PASC.3200:
ALIGN(PARMPTR,LLC1);

Replace P.531 by:
IF IDTYPE^.FORM > POWER THEN

Insert after PASC.3204:
IF VKIND = ACTUAL THEN
BEGIN

Insert after PASC.3207:
END;

UPDATE2:

Replace P.122 by:
FLC := L + K - (K + L) MOD K

Replace P.528 by:
CSTPTRINX := 0;
TOPNEW := LCAPTERMARCKSTACK;
TOPMAX := LCAPTERMARCKSTACK;

FEATURE IMPLEMENTATION NOTES

INTERIM REPORT - IMPLEMENTATION OF SETS - 1

This report addresses results of set implementation tests on three compilers, and a personal estimation of optimal treatment (not yet achieved anywhere so far).

Professor A.H.J. Sale,
University of Tasmania.

Interpretation	CDC6000	B6700	ICL 1900	optimal
set of char	compile error	compile error	allowed, run-check	allowed, correctly
set constructor with value > setmax	allowed	compile error	compile error	compile-error if genuine, however limit should be big, say 256.
set-type constraints	0..58	0..47	0..47, but only checked at runtime.	no limits, except range in reasonable size (256 bits?)

Note: In the ICL compiler, sets may be declared of any subrange type, and the run-time system will be correct as long as no element with a representation outside 0..47 is involved. If this occurs, an "index error" is raised. (I believe this to be liable to lead to undetected and hibernating bugs.)

MACHINE-DEPENDENT IMPLEMENTATIONS

Burroughs B1700

PASCAL ON BURROUGHS B1700
=====

Dear Mr. Mickel,
we have developed a Pascal System for the Burroughs B1700/B1800 Series. The System like many others is derived from the Pascal-P Compiler developed by Wirth and Amman at the ETH-Zuerich.
A preliminary Version has been distributed to several european Universities about a year ago. The System is also the subject of a PhD Thesis in german.

Unlike other B1700 Pascal systems, ours is implemented on top of the B1700 SDL-S Language

Implementation Notes

which also serves as the Basis for the master control program and the utility software. The system runs on MCP-Release 6.0 and higher and is particularly suitable for small machine configurations.

In order to remain compatible with the standard SDL-Architecture only emulation of rearithmetric is provided.

Current Projects include addition of mathematical functions and the design of an "ideal" Pascal architecture.

The system has recently been redesigned and we will gladly distribute it to universities sending us a tape. (We would appreciate tapes in a reusable box. Installations should also indicate if they have use of SDL- and MIL-Compilers). Unfortunately we can not guarantee an errorfree system but we will eventually fix errors made known to us.

Another Pascal-system was produced at our installation by Mr. K. Hausermann. It uses a separate interpreter which emulates the hypothetical stack computer by Wirth and Ammann.

Pascal systems for the B1700 have also been developed at many other universities (Karlsruhe, Newcastle, Dublin, Edinburgh....).

Yours sincerely

Peter U. Schulthess
Institut fuer Informatik
Universitaet Zuerich
Kurvenstrasse 17
8006 ZUERICH
=====
SWITZERLAND

Burroughs B4700 (Fredonia)

George Golden, Sr. (Computer Center; SUNY Fredonia; Fredonia, NY 14063; 716/673-3393) wrote on 78/4/10: "We are trying to get Pascal running on the Burroughs B4700. It runs! But it takes too much core."

Burroughs B6700 (Tasmania)

The PASCAL compiler for Burroughs B6700/B7700 systems written at the University of Tasmania is now available for distribution. To acquire a copy, fill out the attached forms and send to:

PASCAL Support,
Department of Information Science,
University of Tasmania,
Box 252C, G.P.O.,
HOBART, 7001

The compiler is distributed on 9-track magnetic tape, (but 7-track is also available) and an installation manual is supplied, together with two copies of the user-documentation. At present this comprises:

- * Report R77-1 - a supplement to Wirth's User Manual.
- * Report R77-3 - a Reference Manual similar to B6700 Algol's.

- * a PASCAL Language Card.
- * a PASCAL System Card.

Further copies of the user-documentation may be available at production cost.

The charge for the system is Australian \$100 annually, and will be invoiced to you when you receive the tape. The tape remains our property, and should be returned when you have copied its contents so that future releases can be mailed to you. The service will cover:

- * mailing and processing costs,
- * extensions and revisions, and
- * the costs of an FTR-reporting service and maintenance.

Each installation will be issued with FTR-forms similar to those used by Burroughs for use in corresponding with us, and we will attempt to do a professional job in maintenance of the system.

The Tasmania B6700 PASCAL compiler is a true compiler for the B6700 or B7700 computer systems: it generates executable code-files which are accepted by the operating system. Its compilation and execution performance is within a 20% margin of comparable compilers in the B6700 system for average programs. The current version generates LINEINFO in the code-file, but does not generate BINDINFO, so PASCAL programs cannot yet be linked to other code-files. The compiler itself is written in B6700 Algol, as are most of the extra intrinsic procedures it uses.

Objectives of this project were to develop a compiler which enforced compliance with the standard definition of PASCAL as far as possible by utilizing the special features of the B6700 system, while making it a fully integrated member of the B6700 compiler set. These targets have been largely met, and a wide variety of checks are available to the user-programmer; probably to a higher degree than most other PASCAL compilers. However, file attributes, record-oriented formatted i/o, random-access i/o, and compiler options, are provided in a way that will ease the learning problems of existing B6700 programmers. The compiler permits use in a very similar manner to the well-know compilers (Algol, FORTRAN, COBOL, etc).

The compiler has been stable in code for some time, reflecting its basic integrity. However new features are added from time to time, and notified to recipients as patches or as new version releases. The Department accepts FTR notices, and will attempt to fix those which warrant such attention. Some modifications have taken place as a result of user feedback. The compiler was especially designed so as not to generate dangerous code to the MCP, and no system crashes have been attributed to it since the first few months of testing, and then only three!

User-level documentation is provided for the compiler in the form of cards and reference manuals. The standard of these is similar to that of Burroughs' manuals and cards. Systems documentation is more sparse, and consists of some implementation notes, the compiler itself (a microfiche listing is provided), and a report on aspects of the language.

The compiler is in daily use by students at the University of Tasmania.

I must apologize to those of you who wrote enquiring about the availability of our B6700 PASCAL compiler earlier and did not receive a prompt reply. The end of the academic year and a number of important decisions interfered to prevent us making the compiler available as soon as we would have liked.

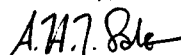
To cut a long story short, the B6700 PASCAL compiler developed at the University of Tasmania is now available from us. There are three conditions:

- (1) each recipient must agree not to disclose the compiler to other parties, and must agree not to supply copies to other institutions.
- (2) an annual fee of \$100 (Australian) is required to cover mailing, processing, and other maintenance charges, payable to "The University of Tasmania".

The compiler has been operational in a student environment at the University of Tasmania for a year and has proved stable and reliable; it has been released on a restricted basis to two other sites for about eight months with similar results. The compiler is provided with a Reference Manual and a Supplement to the User Manual (of Jensen & Wirth), and with ready-reference cards. Recipients are granted copyright permission to reproduce these for their own purposes, and in some cases additional copies may be ordered from the University of Tasmania. The service provided includes the provision of updated versions of the compiler at intervals, and the maintenance of an FTR-service similar to that of Burroughs.

If you want further information before ordering the compiler, please write and we can send you documentation and listings. If you want a copy, please arrange for the non-disclosure notice (FORM A) to be signed by a responsible officer of your institution and the computing centre manager (if applicable), and forward it with the supplementary information (FORM B) to the address given.

Yours sincerely,



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

CII 10070, IRIS 80 (Paris)

0. DATE/VERSION. 78/02/21.

1. Distributor/implementor/maintainer:

implementor:	distributor/maintainer:
D.Thibault	P.Maurice
17, rue Gay-Lussac	Université Paul Sabatier
F-75005-Paris	Informatique
	118, route de Narbonne
	F-31077-Toulouse-cedex
	(61)53-11-20(300)

2. Machine: CII-10070, CII-HB-IRIS 80, XDS-Sigma 7

3. System configuration: Siris7, Siris8. Easily available on other systems: adaptation of run-time routines and perhaps of the code-generation phase of the compiler.

4. Distribution: source programs (Pascal and assembly code), object programs and load modules available on magnetic tape (9 tracks, 1600bpi); send a mini-tape to distributor; mailing cost only.

5. Documentation: user manual, in french (sept. 75); separate papers describe extensions and differences with the User Manual and Report (K.Jensen, N.Wirth); not machine retrievable.

6. Maintenance policy: bug reports are encouraged; announcements of releases are sent to users, together with listings of modifications (errors and/or extensions). Release 5 has been issued in Jan. 78.

7. Standard:

- not implemented:
 - type T= <type identifier>
 - record ... case <type id> of ...
 - (tag field is mandatory)
 - structured types of files.
- extensions:
 - separate compilation
 - VALUE part for global variable initialization
 - heap management through NEW/DISPOSE or NEW/RESET
 - standard files TTYIN, TTYOUT used for interactive applications programming
 - compiler options (source listing, run-time checks, post mortem dump, pseudo-assembler listing of generated code).

8. Measurements:
- compilation space: minimum 32K words; 40K words to compile the compiler.
 - compilation speed: ≈ 2100c/s (Fortran: ≈ 1300c/s)
 - execution speed: programs from N.Wirth (ETH Zürich, March 76):

	Pascal run-time checks	Pascal no checks	Fortran
palindromes	4260ms	3860ms	2970ms
powers of two	1530ms	1470ms	3867ms
prime numbers	1900ms	1700ms	941ms
count characters in a file	5100c/s	5800c/s	5100c/s

9. **Reliability:** good; used since 1974 in 25 installations, mainly for teaching programming and compiler writing, and also for the development of large system software projects.
10. **Development method:** fully bootstrapped from Amman's CDC compiler; generates code for the CII Tink editor; all operating system dependencies are located in a monitor (*2000 lines of assembly code), which must be linked with user programs. The compiler takes advantage of the separate compilation system: it consists of four overlaid modules (*8500 'pretty-printed' Pascal lines). The bootstrap process took about 2 man/years, to produce a compiler for the first version of the language (Wirth 71); adaptation to standard took about 6 man/months.
11. **Library support:** a system library contains the standard Pascal functions SIN, COS, ... and the Pascal monitor (see 10). Separate compilation allows using private libraries, written in Pascal or in any other language; interfacing with other languages requires a knowledge of the compiler. Programs are manipulated under control of a 'Pascal programming system', which provides the users with powerful editing functions, ranging from source inclusion to program transformations. Also provided are interactive debugging at compile and execution time, and library management. The system is entirely written in Pascal (*22000 lines).

Commodore 6502.

Formerly MOSTEK. See DEC LSI-11 (San Diego).

Computer Automation LSI-2, 4

Bob Hutchins (Computer Automation; 18651 Von Karman; Irvine, CA 92713; 714/833-8830x335) wrote on 7/8/3/1: "We just recently brought up sequential Pascal on our new 16-bit minicomputer series, the Naked Mini-4 series. It runs on all models including the NM-4/10 which sells for \$645 including CPU, 4K RAM, and 4 I/o ports. As far as I know, this is the lowest priced minicomputer system that supports Pascal. Our Pascal is based on sequential Pascal supplied by Brinch Hansen. It is supplied at a one time fee of \$900 including compiler, interpreter, and documentation."

Minicomputer News reported on page 2 of their Jan. 5, 1978, issue that "Pascal software (on the LSI-4 line), formerly priced at \$900, will be offered without charge."

Data General ECLIPSE (San Bernardino)

MEDICAL DATA CONSULTANTS

March 10, 1978

1894 Commercenter West, Suite 302, San Bernardino, CA 92408

Dear Andy,

We have spent the last several months in a reconsideration of our entire PASCAL endeavor. As we reported previously, we have developed a new version of Data General compatible PASCAL which is significantly faster than our previous version, but which continues to use a 64-bit data path, is fully RDOS compatible and easily modifiable and extendable. We had previously intended to take this version to market as a low priced, but profit making venture, as reported in the February PASCAL NEWS.

As part of our continuing PASCAL development we now have a preliminary implementation of a PASCAL compiler which produces code that executes at the speed of that provided by Data General's Optimizing FORTRAN 5. We expect, however, the full development of this product will take 6-12 months.

We have decided, therefore, to release our current improved version of Data General PASCAL for a reproduction cost of \$100.00 on 800 BPI, 9 track magnetic tape. This includes executable object code, source code and machine readable documentation.

Please find attached a standard description of the product.

Sincerely,



Ted C. Park
Director, Systems Development

DISTRIBUTOR/IMPLEMENTER/MAINTAINER

Ted C. Park
Director, Systems Development
Medical Data Consultants
1894 Commercenter West
Suite 302
San Bernardino
CA 92408

MACHINE

Data General - any ECLIPSE-line computer

SYSTEM CONFIGURATION

ECLIPSE must have FPU or EAU
Minimum of 16K words user memory
RDOS REV 6.1 or 6.2
FORTRAN 5 (any recent revision)

DISTRIBUTION

System supplied on 9-track 800 BPI tape in RDOS 'dump' format. The cost is \$100.00 to cover our mailing and duplicating costs.

DOCUMENTATION

User must obtain his own copy of the Pascal Users Manual and Report. It is recommended that the user obtain an implementation kit from the University of Colorado. Documentation and operating procedures are supplied on the tape.

MAINTENANCE POLICY

Bug reports are welcome but no formal commitment for support can be made at this time. To date, no bugs have been reported.

STANDARD

PASCAL P4 subset



(714) 825-2683

MEASUREMENTS

Compilation Speed:	50 chars/sec (including blanks and comments)
Word Size:	64 bits
Real Arithmetic:	Uses 64 bits
Integer Arithmetic:	Uses 32 bits
Execution Speed:	Fairly slow (since it is interpreted!)
Minimum Memory Needed:	16K words
Virtual Memory Required:	A contiguous disc file of 524,288 bytes

RELIABILITY

Version 1 exists in at least 10 sites, we believe no bugs exist. Version 2 is primarily the same as Version 1 except with improved operating procedures, faster compiles and executions, and increased capability. As such we anticipate few, if any, bugs.

DEVELOPMENT METHOD

Developed from PASCAL-P4. P-CODE assembler and interpreter written in assembly language. All programs are extremely modular and well documented so that any changes wished by the user should be easy to incorporate.

LIBRARY SUPPORT

No Data General libraries are needed to run the system nor is it possible to use any if desired.

Data General NOVA, ECLIPSE (Columbia)



Rhitek, Inc.

Computer Engineering

P.O. Box 220, Columbia, Maryland 21045

March 8, 1978

Dear Andy,

RHINTEK, Inc. is making available its PASCAL compiler to other Data General NOVA/ECLIPSE users. This compiler is used by RHINTEK as an application and system programming language and will continue to receive support and enhancements by us. We are using the compiler on a NOVA 3/D running Rev. 6.10 mapped RDOS. However, we are cleaning up the code and expect the compiler to be able to run under unmapped RDOS on a 32K NOVA within a few weeks.

Below is the checklist information on our PASCAL compiler for Data General NOVA (or equivalent) computers.

DISTRIBUTOR/IMPLEMENTOR/MAINTAINER -- RHINTEK, INC; Box 220; Columbia, Md. 21045 (301)

MACHINE -- Data General NOVA or ECLIPSE minicomputers or equivalents.

SYSTEM CONFIGURATION -- Mapped RDOS system or 32K unmapped RDOS with minimum operating system. The current revision of Data General RDOS will be supported but the compiler should work with older levels.

DISTRIBUTION -- 9 track magnetic tape, 800 BPI, 7.5 inch tape in the RDOS dump format. Price for a single user license is \$975. Multi-use, OEM's, and educational licenses will be handled on a separate basis.

DOCUMENTATION -- The package includes source code, binary code, and ready to run demo programs. Instructions for executing the compiler are included; the operational information can be obtained from the books by Per Brinch Hansen and Al Hartman.

MAINTENANCE POLICY -- Updates for one year and notification of substantial enhancements as long as interest is shown. We will maintain a users group and encourage bug reports and suggestions.

STANDARD -- Based on Sequential PASCAL written by Per Brinch Hansen and Al Hartman. The current version lacks: "file, goto, label, and packed" reserved words and sqrt, sin, cos, arctan, ln, exp, sqrt, eof, eoln, odd, and round built in functions. This is a seven pass Sequential PASCAL compiler written in PASCAL and generating code for a hypothetical 'stack' machine. The code is interpreted by a program written in NOVA assembly language.

MEASUREMENTS -- The compiler compiles source code at the rate of 200 line/min. This is about one-half of the rate of the PDP 11/45 but five to ten times the speed of the other compilers on the NOVA. The compiler will compile itself in about 30 minutes total.

RELIABILITY -- good

DEVELOPMENT METHOD -- The virtual machine interpreter was coded in NOVA assembly language and then the compiler was modified along with interpreter into its present form.

LIBRARY SUPPORT -- There is no library support as yet. The operating programs support program swapping or chaining with only minor effort as this is used with the compiler.

Sincerely,

Rainer McCom

Rainer McCom, President
Rhitek, Inc.

A package of UNIX software is available from the Computer Science Division of the University of California at Berkeley. This package includes the instructional Pascal system which has been in use at Berkeley this past year and the standard Berkeley editor ex, an extension of the standard UNIX editor ed which offers many new and improved features. The Pascal system requires separate I/D space to run (an 11/45 or 11/70); ex will run without separate I/D but requires a full load of user core (64 bytes).

UNIX Pascal is designed for interactive instructional use. It produces interpretive code, providing fast translation at the expense of slower execution speed. An execution profiler and Wirth's cross reference program are also available with the system. The system supports full Pascal, with the exception of procedure and function names as parameters. The language accepted is very close to 'standard' Pascal, with only a small number of extensions for the UNIX system. (An option restricts the implementation to the standard.)

The UNIX Pascal User's Manual gives a list of sources relating to the UNIX system, the Pascal language, and the UNIX Pascal system. Basic usage examples are provided for the Pascal interpreter components pl, px, and pxp. Errors commonly encountered in these programs are discussed. Details are given of special considerations due to the interactive implementation. A number of examples are provided including many dealing with input/output. An appendix supplements Wirth's Pascal Report to form the full definition of the UNIX implementation of the language.

Source code, binaries and machine readable versions of all documentation are included with the tape. The Pascal system and the ex text editor are distributed under a license agreement; UC Berkeley is thus the sole source for this software. The software is distributed only to UNIX licensees and only for non-commercial purposes. A copy of the cover page of the UNIX license agreement is an acceptable form of proof of license.

The distribution tape is a standard "tp" format, 800 RPI magnetic tape. A 1200 foot reel is the minimum and preferred size. There is a one time \$50 charge (\$65 for overseas airmail) for a copy of this tape. This charge includes the costs of preparing the tape, mailing costs, and the costs of distributing future updates and corrections to the programs and documentation on the tape. These updates and corrections will be distributed at regular intervals as their volume and severity warrants. Also included with the tape are high quality copies of the UNIX Pascal User's Manual and the Ex Reference Manual which require a phototypesetter to produce. It is also possible to obtain a copy of the documentation without getting a copy of the tape. The \$5 charge for this copy may be deducted from the tape charge if you later decide that you want a tape. If you prefer, you may send an additional \$10 and we will purchase a tape on which to send you the software.

To receive a copy of the license agreement (which must be signed before you can receive the tape) write to:

Berkeley UNIX Software Distribution
c/o William N. Joy
Computer Science Division
Department of EECS
Evans Hall
University of California, Berkeley
Berkeley, California 94720

Questions about this tape can be directed to William Joy at the address above or at (415) 642-4948. Messages can be left at the Computer Science Division office phone (415) 642-1024.

0. DATE/VERSION. 77/12/22.

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Seved Torstendahl, TN/X/Tda, Telefon AB LM Ericsson, S-126 25 Stockholm, Sweden, tel 08-719 4909.

2. MACHINE. Running on PDP-11 model 35 and up and generates code for all PDP-11's. Crosscompiler on DEC-10 generates code for all PDP-11's. The compilers generate code for floating point hardware and extended arithmetic if option switches are set.

3. SYSTEM CONFIGURATION. RSX-11M or IAS. (DEC-10 crosscompiler under TOPS-10). PDP-11 with memory management and a user partition of at least 28 Kwords, preferably 32 Kwords. It would be an easy task to replace the RSX interfacing routines with new ones interfacing DOS or RT-11. We don't plan to do that work.

4. DISTRIBUTION. The compilers are available at \$50, plus \$10 if we supply the tape (600 feet). 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 in one or more of the following formats, indicate which you want:

- three DECTapes in PDP 11 DOS format (DEC10 and PDP11 users)
- one 9-track magnetic tape in DEC 10 format (DEC10 users)
- one 9-track magnetic tape in industry compatible format (users of DEC10 and other computers)
- one 9-track magnetic tape in DOS format (PDP11 users).

5. DOCUMENTATION. A machine retrievable user manual complementing the PUM&R book is included on the distribution tape.

6. MAINTENANCE. No responsibility, but if errors are found reports will be distributed to known users. Error reports and improvement suggestions accepted.

7. STANDARD. Restrictions:

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

Extensions:

- function results can be of nonscalar type;
- arrays with unspecified bounds (but specified index structure) can be used as formal parameters to procedures, allowing differently declared variables or constants as actual parameters;
- a string parameter type has been 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 declaration with the procedure block replaced by "extern";
- most option selectors ((**M)*) etc.) are settable by switches in the MCR command line (version 5 December -77).

8. MEASUREMENTS. The compiler requires a 32 Kwords partition (at least 26 Kwords for small programs). Compilation speed is about 300 char's per second. In a 64 Kwords partition using PLAS under RSX-11M increases speed to about 3000 characters per second.

9. RELIABILITY. Excellent. The compiler is now in heavy use at five sites, and is distributed to 17. Only minor errors have been found since July -77. First version released April -77. Latest version December -77.

10. METHOD OF DEVELOPMENT. The crosscompiler for PDP-11 running on DEC-10 produced by Bron et al was used as input. It is written in Pascal and developed from Pascal-P. This compiler was modified to generate object code linkable under RSX/IAS and to give access to the file system of RSX/IAS. When the crosscompiler was finished it compiled itself and transferred to PDP-11. The implementation effort until now is about 7 manmonths.

11. LIBRARY SUPPORT. Separate compilation allowed. Possible to use external procedures written in FORTRAN (or assembler). The December -77 version also gives: Automatic copy of text from library into source program ("include"); execution frequency measurements; execution trace; option selectors (\$R- etc) settable by switches in the MCR command line. Next version (spring -78) will also include a symbolic post-mortem dump and an interactive source-level debugging package (mainly copied from DEC-10 Hamburg compiler).

DEC PDP-11 (OMSI) (formerly ESI)

0. DATE/VERSION. 77/12/26; "OMSI Pascal-1" (formerly "ESI Pascal").

1. IMPLEMENTOR/DISTRIBUTOR/MAINTAINER. Oregon Minicomputer Software, Inc. (OMSI); 4015 SW Canyon Road; Portland, OR 97221; 503/226-7760. Implementors: John Ankcorn, Don Baccus, and Dave Rowland.

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

3. SYSTEM CONFIGURATION. Minimum of 16K words. Operates under RT-11, RSTS/E, or RSX.

4. DISTRIBUTION. Compiler, support module, cross referencer, text editor and instruction manual available for \$1500 (\$995 for educational use). Available on 9 track 800 bpi magnetic tape, or DEC cartridge disk.

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 I/O 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 40% 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. Over 60 installations, and growing steadily.

10. DEVELOPMENT METHOD. Single pass recursive descent compiler written in Macro-11. Hand-coded based on University of Illinois bootstrap (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 VAX-11/780

0. DATE/VERSION. 78/03/27.

1. Implementor/Distributor/Maintainer.

Implementor: Professor Hellmut Golde
Department of Computer Science
University of Washington
Seattle, WA 98195
Tel. 543-9264 (Area Code 206)

2. Machine: DEC VAX-11/780 in native mode.

3. System Configuration: DEC VAX-11/780 under the VMS Operating System.

10. Development Method:

The compiler will be derivative of the CDC 6000/CYBER compiler.
The compiler will be transported to the VAX system via cross-compilation.

Hewlett Packard HP-2100, 21MX (Trieste)

Paolo Sipala (Istituto di Elettrotecnica; Universita di Trieste; Via Valerio, 10; 34127 TRIESTE; Italy) wrote on 78/03/20:

I have recently completed a Pascal-S compiler/interpreter for the HP 2100/21MX computer, running under DOS-IIIB. I enclose the Documentation Form accompanying the submission of the program to the Hewlett-Packard Software Center, Contributor Section (11000 Wolfe Rd.; Cupertino, CA 95014), through which the program should be available for distribution in the near future.

To summarize the data in the form, the system requires a 11K main core area (so it might fit into a 16K system, if the resident DOS modules are kept to a minimum, but 24K is more comfortable); there are separate versions for non-EAU, EAU, and floating point options machines. It is not noticeably slower than the standard compilers while compiling, and not worse than the standard interpreter (Basic) while interpreting. It has been subjected to rather limited testing (a few dozens programs from the Pascal Manual) and is being now offered to students here for their exercises.

Until the program becomes available through HP Software Contributors Center, I might send a copy of the program to those who request it by enclosing the price of the mailing (the weight is about 2 lbs.).

Hewlett Packard HP-3000

0. DATE/VERSION. 78/04/15.

1. DISTRIBUTOR: The system is available in the HP-3000 Contributed Library, Volume 4. Contact your local sales office, or write:
Hewlett-Packard Company
Contributed Software
P.O. Box 61809
Sunnyvale, CA 94088

IMPLEMENTOR: Robert A. Fraley
Hewlett-Packard Laboratories
3500 Deer Creek Rd.
Palo Alto, CA 94304

MAINTAINER: Maintenance is not provided, but errors may be mailed to the implementor.

- 2. MACHINE: HP-3000.
- 3. SYSTEM CONFIGURATION: MPE.
- 4. DISTRIBUTION: The system will be available through the HP-3000 Contributed Library in June, 1978.
- 5. DOCUMENTATION: Sparse machine-readable documentation is included.
- 6. MAINTENANCE: None. Error reports may be sent to the implementor, and may be fixed in later releases. Full file support and separate procedure compilation may be available in a future release.
- 7. STANDARD: Falls short of the standard due to the sorry state of the P-compiler. Measures are being taken to improve the P-compiler.
- 8. MEASUREMENTS: No specific measurements made. Some improvement will be available in a future release. The compiler is somewhat awkward to use, due to the P-code intermediate. Compilation and link-edit of the compiler operates at 125 lines per minute.
- 9. RELIABILITY: Good. Currently in use at nine installations.
- 10. DEVELOPMENT METHOD: Bootstrapped from a P-compiler by Grant Munsey, Jeff Eastman, and Bob Fraley. Compiles to HP-3000 machine code.
- 11. LIBRARY SUPPORT: None yet.

IBM 360/370 (Australia)

Cox/Tobias letter(s).

AUSTRALIAN ATOMIC ENERGY COMMISSION

NUCLEAR SCIENCE AND TECHNOLOGY BRANCH

RESEARCH ESTABLISHMENT, NEW ILLAWARRA ROAD, LUCAS HEIGHTS

TELEGRAMS: ATOMRE, SYDNEY
TELEX: 24562
TELEPHONE: 531-0111

ADDRESS ALL MAIL TO:
AAEC RESEARCH ESTABLISHMENT
PRIVATE MAIL BAG, SUTHERLAND 2232
N.S.W. AUSTRALIA

IN REPLY PLEASE QUOTE: JMT,mwb

13 March, 1978.

Dear Andy,

Just a note to let you know the current status of Pascal 8000 for IBM360/370 computers.

We are currently distributing version 1.2 of the system. The differences between 1.2 and our earlier 1.1 distribution include a few bug fixes (there were some installation problems on VSI), and a few new features, such as the inclusion of the characters `_`, `[`, `]`, `&`, `|` and `~`. We are very happy with the reliability of the system, - this too has now gone from very good to excellent. We very much enjoy the reports received from Hal Perkins at Cornell University. His letters to us are somewhat overwhelming (average - 10 pages), and we really appreciate his feedback. We only wish more sites would drop us a note as to their progress.

We have now shipped a system to Judy Bishop at the University of the Witwatersrand, and we enjoy corresponding with her. We hope that Pascal 8000

will meet all of her expectations, and we look forward to hearing her comments on the system.

Judy passed on your thoughts of setting up an American distribution centre; we somehow feel that this may cause more problems than it is worth. We cannot understand why people in the U.S. are afraid to contact us directly - perhaps they doubt that Australia has an adequate postal system (no, letters are not delivered in the pouches of kangaroos). We had some delays in the processing of orders earlier in the year, mainly due to our deciding to drop the non-disclosure agreement. We have since established a rather smooth distribution setup and have involved another person to handle the answering of correspondence, mailing of tapes, etc. We answer all initial enquiries with an order form and a copy of our Reference Manual; on receipt of the order form, we despatch the system and invoice the organisation for \$A100 at a later date. The time from our receiving an order form to despatching a system should be no longer than five days, i.e. the system should be in the recipient's hands three to four weeks after they post their order, provided there are no unforeseen delays. We feel that this is not an unreasonable period of time.

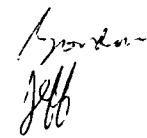
We see a number of problems arising if we were to establish alternative distribution centres - who supplies and copies the tapes, who prints the manuals, who fixes the bugs, who answers technical questions, who supplies the updates, and so on. We are, however, willing to hear of any strong arguments supporting such a centre.

We now have 40 Pascal 8000 sites operational; those on Version 1.1 automatically received the updates to bring them up to 1.2. We anticipate more orders as a result of our dropping the non-disclosure agreement. We are planning a Version 2, but cannot anticipate its release.

We have sent a copy of our latest Reference manual to you under separate cover to add to your undoubtedly desk-high pile of manuals. We hope you find it of interest.

And finally, let us say how much we appreciate your efforts in the Pascal Users Group, and your words of encouragement for Pascal 8000.

Best regards,



Gordon Cox
Jeffrey Tobias
Systems Design Section

Intel 8080 (Ann Arbor)

Jim Rogan (Comshare; Wolverine Tower; 3001 S. State St.; P.O. Box 1588; Ann Arbor, Michigan 48106; 313/994-4800) wrote on 78/2/16 that Comshare "can currently cross-compile [Pascal] source for the Sigma 9 and an INTEL 8080 machine."

The following is an overview of COMSHARE'S PASCAL compiler system. It is presented and outlined with respect to a "package" that could be delivered, from which you could implement the system on your machine.

I. History

Comshare's PASCAL compiler was originally a bootstrapped version of the portable Pascal 'P' compiler. The impetus for the compiler project was to provide the company programmers with a state-of-the-art language from which they could write readable, easily maintainable, efficient programs. Along with these objectives, machine independent programs were sought and this feature was designed into the compiler system. It was decided that the portable PASCAL compiler, with some major modifications would be a reasonable base to start from.

II. PASCAL Language Modifications

In areas where the language definitions were found undesirable or inadequate, modifications were made. The areas primarily effected were the I/O and scoping structure. In brief, the standard INPUT and OUTPUT files were eliminated along with the GET and PUT operations. They were replaced with 'FILE' declaration types, OPEN and CLOSE primitives. The READ and WRITE statements were modified and binary file operators were added.

Also, the scoping mechanism was eliminated (ie. all procedures are considered on the same "level") because it was contrary to structured programming principles, allowing for pathological data references, etc. All the basic language statements, control structures and the declaration sections are the same or enhanced.

Note: Since the language has become an off-color PASCAL, the name has been changed to PASTEL.

III. Operational Characteristics

A. System design

Comshare's PASTEL compiler is a three phase (pass) language processor system. The first phase is a machine independent phase, the second and third are target machine dependent. The process basically consists of translating a source program into a machine independent intermediate form of code for a hypothetical stack computer (see "The PASCAL <P> Compiler: Implementation Notes", NORI, AMMANN, JENSEN, WIRTH). Then, for any given machine, a code generator for it converts the intermediate code into hard machine code.

The first phase (compiler) has three functions: to syntactically analyze the source program; to translate the program into a form of "assembler-like" intermediate instructions (P-codes) and directives; and finally to perform the static and dynamic data allocation.

The second phase also has three discrete functions: to translate the P-codes into a form suitable for code emission (triples) and optimization; optimization, and the emission of the machine instructions themselves.

The third phase is necessary for portability purposes. It is simply running the target machines assembler over the generated symbolic instruction to produce a loader compatible relocatable binary object file. The process can be viewed as follows:

Please note that for the ultimate "production" compiler, one would want to eliminate the third phase by adding a module to the code generator to emit relocatable binary directly. The emission of the symbolic meta symbol could then be an "optional" feature for the compiler to aid in analysis and debugging of the systems you apply this language to.

IV. Compiler Specifications and Limits

Aside from our current, and most highly recommended compiler, we have available two predecessors from which it evolved. A list of pertinent facts relating to each version follows. All timing estimates are based relative to our XDS 1968 FORTRAN compiler which is a one pass processor written in a low-level language.

A. PASCAL THREADED -CODE INTERPRETER

This version implements the language essentially as described in Jensen and Wirth's User Guide / Report.

- uses the ETH character set.
- no external procedures.
- generates macro's that are assembled into threaded calls to runtime interpreter.
- very limited I/O facilities.
- do not know the specific core requirements but I'm sure it's no problem.
- runs at approx. 10.0 times the speed of FORTRAN.

B. PASCAL COMPILER

This is a "real" compiler in the sense that all interpreter functions were eliminated and replaced with a code generation phase. The general enhancements are as follows:

- uses EBCDIC character sets.
- augmented P-code set.
- I/O still limited but faster.
- full set of data types.
- stack machine operations are simulated in registers where possible.
- maximum core requirement is approx. 40K words (??).
- language complement is very close to "standard" PASCAL.
- runs at approx. 3.0 times the speed of FORTRAN.

C. CURRENT PASTEL COMPILER

This compiler is very close to our version of a finished product. It has a lot of enhancements in the areas of usability, efficiency and machine independence. It contains user-oriented features, a new and optimizing code generator and cross-compile abilities working for a Sigma-9 and an INTEL 8080 micro computer. Its language and feature descriptions can be reviewed in the enclosed preliminary reference manual. They are highlighted by:

- compiler option recognition.
- language processor control program.

- full complement of I/O facilities that are very efficient.
- external non-PASTEL procedure linkage.
- dynamic arrays.
- static and dynamic data allocation.
- packed data structures and data allocation options.
- a manual.
- very good documentation (in English) of the internals.
- 'LOOP' statement.

COMPILER:

source language is PASTEL and is approx. 6600 lines of code + comments; object size is 31K words;

CODE GENERATOR:

source language is PASTEL and is approx. 2900 lines of code + comments; object size is 17K words;

RUNTIME:

source language is meta-symbol with a little PASTEL and is relatively small in size; requires 1.8K words of core for code + storage buffers.

- runs at approx. 1.5 times the speed of FORTRAN.
- good testing procedures for releases of new versions.

V. Implementation Considerations (MACHINE X)

1. Must develop a code generator targeted for your specific machine. This would basically involve modifying the code emission routines within our "skeleton" code generation phase processor.
2. A runtime must be developed to support the emitted calls for I/O and a few miscellaneous functions. The runtime is approximately 90% I/O routines interfacing with the operating system, 6% house keeping routines and the remainder consists of miscellaneous system functions to support language features. These routines could be written in PASTEL and developed concurrently with the code generator using COMSHARE timesharing services or, could be done on your given system in any language desired.
3. The compiler "process controller" will need some minor changes to do the appropriate subprocess start-up, termination and communication control.
4. Modifying the code generator mechanisms to incorporate the new procedure calling protocol for interfacing with non-PASTEL languages.

VI. Implementation Considerations (XEROX SIGMA 9)

1. The compiler and code generator can be directly assembled by the meta-symbol processor, since they are coded in PASTEL.
2. The runtime will need some modifications for interfacing with CPV. These changes should be strictly limited to the I/O interface. Our system does not have a 'DCB' concept and it would be necessary to install these into the runtime to do the physical data transfers. All the other code is in the Sigma-9 instruction set and PASTEL.
3. The process controller will need some re-writes to do the appropriate subprocess startup, termination and communication.
4. Modifying the code generator to incorporate the new procedure calling protocol for interfacing with external, non-PASTEL languages.

Intel 8080 (Munich)

1. Implementors:

D. Krall, W. Remmele, U. Weng
Siemens AG
ZT ZFE FL SAR 121
Otto-Hahn-Ring 6
D-8000 München 83
Germany

2. Machine:

Intel MDS800 (under ISIS II) with 8080 processor;
Host-machine: Siemens 4004/151 (or any with a Pascal-system)

3. System configuration:

64 K Byte RAM, Floppy Disk, Console;
A possibility to transport the intermediate code from the host-computer to the MDS.

4. Distribution:

No final decisions made yet - contact implementors.

5. Documentation:

A manual is available (written in German). Updating is done with each new version.

6. Maintenance:

No final decisions made yet.

7. Standard:

No changes to the Standard. The attribute packed is ignored.
Current restriction: No functions as parameters.
Extension: external procedures.

8. Measurements:

No measuring has been done yet.

9. Reliability:

Seems to be excellent: No known errors.

10. Development method:

Compiler derived from ETH's P4; new Assembler, Linkage-Editor and Interpreter. A resident version for the MDS800 is in work.

Marinechip Systems
computer hardware and software
16 Saint Jude Road
Mill Valley, Ca. 94941
(415) 383-1545

March 22, 1978

Timothy M. Bonham
D605/1630 S. Sixth Street
Minneapolis, MN 55454

Dear Tim:

The many extended conversations that went on at the Computer Faire resulted in some scrambled information being received. The Interdata 7/16 Pascal compiler that I have a copy of is the cross-compiler for the Univac 1100 that was done by Mike Ball of the Naval Ocean Systems Center (formerly Naval Undersea Center) in San Diego. His compiler is a version of the Hartmann / Brinch Hansen compiler with the interpretive code generation pass removed and three phases added which generate Interdata machine code. He has both the Sequential and Concurrent compilers running (with common code generators), and an Interdata kernel for Concurrent Pascal. The compiler was written with "source code configuration" statements in it so that either a Univac or an Interdata version can be generated by processing a common source with a Pascal program. As of the time I got a copy of the compiler (about a year ago), only the cross-version was running, and the bootstrapping to the 7/16 was not yet complete. I have not talked with Mike to find out whether the compiler is yet running on the 7/16 itself. I do know that the Univac version was producing workable 7/16 code.

I understand that Mike now has the Interdata 8/32 version compiling itself on the 8/32. Apparently the 8/32 version is extended beyond the original 7/16 design, and may be moved back down to the 16 bit series. In any case, the person to contact about all this stuff is Mike, not me. (Mike is a PUG member, and his address is listed in the roster).

I got a copy of Mike's compiler in the hopes of using it as a base to build a true compiler for the T.I. 9900 machines I am building. At present, we are taking a hydra-headed approach to Pascal. We are looking at the UCSD Pascal, and also at bootstrapping the original Concurrent Pascal via the interpretive code. Once we have a workable interpretive Pascal, we will do the true compiler if we feel the need.

I hope this information has been of use. I will send in an implementation checklist for my 9900 Pascal as soon as it is running.

Sincerely,


John Walker

Northwest Microcomputer Systems 85/P

Northwest Microcomputer Systems (121 East 11th; Eugene, OR 97401; 503/458-0626) is marketing an Intel 8085A based system which supports UCSD Pascal — see DEC LSI-11 (San Diego). Hardware includes two floppy disks (1 megabyte), 54K bytes of 450ns static RAM, a keyboard, 24 by 80 char CRT, 2 serial ports, and several parallel ports. The price is \$7495. Also included is the CP/M operating system.

Prime P-400

0. DATE/VERSION. 78/03/01. GEORGIA TECH PRIME 400 PASCAL COMPILER.
1. IMPLEMENTOR/DISTRIBUTOR:
Professor Richard J. LeBlanc
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332
2. MACHINE: PRIME 400
3. SYSTEM CONFIGURATION: PRIMOS IV Operating System, 64V mode, 128K bytes minimum.
4. DISTRIBUTION: A first release of the compiler should be available by July 1978. Further details are not yet finalized.
5. DOCUMENTATION: None yet available beyond PASCAL-P documentation.
6. MAINTENANCE POLICY: Error reports from users will be encouraged. Details concerning distribution of corrections and updates not yet finalized.
7. LANGUAGE IMPLEMENTED: PASCAL-P subset of Standard PASCAL.
8. MEASUREMENTS: Not yet available.
9. RELIABILITY: Not yet available. (It is intended that this implementation project will eventually result in a highly diagnostic and very reliable compiler.)
10. DEVELOPMENT METHOD: The code generation parts of the PASCAL-P4 compiler are currently being rewritten to generate PMA calls to interpreter routines. This will then be assembled and linked with those routines, producing a "threaded code" interpretive program. The compiler will be bootstrapped to the PRIME using PASCAL-6000 on a CDC CYBER 70.
11. LIBRARY SUPPORT: None yet available. Support for external procedures written in PASCAL, FORTRAN and PMA will be an early addition to the compiler.
12. FURTHER DEVELOPMENT: As soon as this first version is available, work will begin on adding code generators to produce directly executable code. At the same time, implementation of full PASCAL will be under development. Many of the diagnostic features currently found in the UW-PASCAL compiler for UNIVAC 1100 machines will also be included.

