PASCAL USER'S GROUP

PASCAL NEWSLETTER

NUMBER 7

COMMUNICATIONS ABOUT THE PROGRAMMING LANGUAGE PASCAL BY PASCALERS

FEBRUARY, 1977

TABLE OF CONTENTS

0 POLICY

1 EDITOR'S CONTRIBUTION

3 HERE AND THERE WITH PASCAL

3 News

4 Conferences

4 Books

4 Roster

8 ARTICLES

8 "Life, Liberty and the Pursuit of Unformatted Input"
   - D. W. Barron and J. M. Mullins

9 "Pascal Printer Plotter"
   - Herbert Rubenstein

17 "Yet Another Look at Code Generation for Pascal on CDC 6000 and Cyber Machines"
   - Lawrence A. Liddiard

24 OPEN FORUM FOR MEMBERS

27 IMPLEMENTATION NOTES

27 General Information

27 Checklist

27 Portable Pascals

29 Software Writing Tools

29 Compilers

45 ALL PURPOSE COUPON
PASCAL USER'S GROUP POLICIES

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

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

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

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

In the United Kingdom, send £2.50 to:
Pascal Users' Group/ %Judy Mullins/Mathematics Department/The University/ SOUTHAMPTON/S09 5NH/United Kingdom/ (telephone 0703-559122 x23B7).

PASCAL NEWSLETTER POLICIES

The Pascal Newsletter is the official but informal publication of the User's Group. It is produced quarterly (usually September, November, February, and May). A complete membership list is printed in the November issue. Single back issues are available for $1 each. Out of print: #s 1, 2, 3
#4 available from George Richmond/Computing Center/U of Colorado/Boulder/80309

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

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

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

WRITTEN INFORMATION FOR THE Newsletter IS EASIER TO PRINT IF YOU TYPE ALL MATERIAL 1½ OR DOUBLE SPACED SO THAT IT IS IN "CAMERA-READY" AND "PHOTO-REDUCIBLE" FORM FOR THE PRINTER. REMEMBER, ALL LETTERS TO US WILL BE PRINTED IN THE Newsletter UNLESS THEY CONTAIN A REQUEST TO THE CONTRARY, AN OVERRIDING GUIDE SEEN IN AN OLD MAD MAGAZINE APPLIES: "ALL THE NEWS THAT FITS, WE PRINT!"

If we take an "advocacy" position with respect to Pascal in order to promote its use, these and other aspects come into play. At the University of Minnesota, the following proved very useful since we began using Pascal in 1972.

1) The prevailing view that a language processor should be given support only proportional to its usage cannot be tolerated. To give a language processor a fair chance, it must be actively promoted for awhile and then its acceptance evaluated. The intrinsic merit of the language and its processor should be the determining factors.

2) Proper user documentation helps promote the language. Pascal starts with Pascal User Manual and Report. In addition a local computer center should provide two documents: A) a double-sided one sheet handout describing the local Pascal facilities to satisfy the numerous requests for information made by walk-in users. On this sheet should be a date, place, author, and a description of the purpose of the sheet. The installation's Pascal facilities include references for more information (including P.LUMAR and the other document described below); a description of the compiler or interpreter, its origin, reliability, commonly used options, and how to use the system in both batch and interactive modes (this includes command sequences and a description of the form of compile-time and run-time messages); and a small character set table if substitutions need to be made. B) a larger (20-30 page document) both in printed copy and in machine readable form which includes an introduction describing the scope of the document; information about the programming language Pascal, its history, uses, implementations, general and short description of its semantics and syntax; a history of the particular implementations (Pascal compilers) the computer center is running (current features and future developments); a description of the implementation, how it works, the specific definition of the sizes of scalar types, predefined (non-standard) identifiers, compiler options and switches, differences between this implementation and the standard; how to run programs under the implementation both in interactive and batch modes, program preparation, character sets, the commands to invoke the compiler or interpreter; guidelines, hints and cautions for effective usage, error messages, how to use software writing tools such as the cross-referencer, prettyprinter, source language editor, etc.; a detailed annotated list of references: introductory texts, reference manuals, books on applications, and sources of current information.

3) Enough people must be available for helping users with problems in their programs.

4) Publicity for Pascal to keep it constantly in the eye of the computer center user community: 2-3 week short courses in the language; articles in the computer center newsletter promoting the language as well as announcing planned changes in versions;
EDITOR'S CONTRIBUTION

living, "useful", well written, and simple example programs to show the language at its best: (e.g. a fancy calendar program - what better way to get people's interest?)?  

5) When converting persons remember that: don't waste time converting Fortran and assembly language programmers who are overly concerned with machine efficiency. They will persist in their habits. Pascal's strengths lie in reducing the number of runs one has to make on program development, and because Pascal is nearly as efficient in terms of machine time as Fortran, less actual computer time is used; new programmers are the best bet, the computer science department can make a great contribution by teaching the language to new programmers and using it in other parts of the curriculum; urge people to write new programs in the language rather than getting them to convert old programs - although the latter may produce converts astounded at better solutions arrived at because they were able to think more clearly and conceive of a better algorithm in a systematic language (Pascal). 

After bringing Pascal from nowhere to third out of 20 languages in four years we feel that support for Pascal is sufficient to survive and "ecological counterattack" and will continue to erode Fortran's base of users as we satisfy more of William Walde's principles - particularly in the area of libraries of procedures.

PART II - Pascal and Standards

There has not been time to receive the reaction to the proposals which appeared in Newsletter #6. Formal standardization of Pascal as it is now (by an official standards organization such as ISO or ANSI which could then have economic enforcement in the marketplace) is pretty straightforward. Changing Pascal is certainly a political problem, and even deciding how to pick a committee and when and where it could meet may prove to be overwhelming. A lot of issues regarding specific changes are not clear cut. We are a loose union, not a tight band, of devotees.

What we should concentrate on is conventionalizing the few recurring extensions in the various implementations of Pascal. We can use the Newsletter for that. I cannot overemphasize my conviction that much, careful consideration was given to what features were left out of Pascal as it has evolved. We must always go back to the design goals of compactness, vehicle for portability, vehicle for teaching systematic programming, and a tool to write efficient production programs. We should not use Pascal for purposes it was not intended (such as writing an operating system). One should not misuse or break any tool. Note that other languages have been designed for those tasks (in the case of operating systems with the need to express concurrent processes we have Brinch Hansen's Concurrent Pascal, Hoare's SIMONE, and a rumor about Wirth's MODULA).

I hope my editorial in PUG#6 did not seem too confused - I was trying to be compromising and all-encompassing and I still lack a lot of answers.

One final note: the wholeSALE bending of Pascal to make it conform to conventions of Burroughs ALGOL (as described in the report: "Burroughs Pascal: Some Implementor's Thoughts) is alarming. Why have a different language (Pascal) available which can bridge Burroughs users to software written on other machines if one adopts so many features from an existing Burrough's language: Burroughs Extended ALGOL? One might as well stick to Burroughs ALGOL. At best it's PascALGOL! Why get so upset? Implementors are not operating in a vacuum; they affect all of us on the issue of standards and portability.

PART III - PUG and Pascal Newsletter

PUG now has 598 members in 24 countries and 44 states. We have been growing steadily at the rate of 60 members/month since we started. By June, then, we should have almost 1000 members. This will cause us financial anguish in the form of growing pains. So the result is very poor service regarding back issues. Sorry. It's hard to plan ahead with the small budget we have.

Speaking of slow distribution, as I write this I'm sure some of you have not yet received #6. This will make #8 longer when the reaction arrives. #5 and #6 were big and were mailed overseas by air so that we could get the Pascal movement back on track. We also printed nearly everything that came to our attention. This satisfied the individual urgent questions we had been receiving about newsletters and implementations. This did not really help, but it did get us back on track. We have tried to take an advocacy position with the goal of furthering Pascal. We hope that this hasn't offended anyone.

The bright area is the UK distribution center which is getting lots of members and has eased the overseas distribution load for us.

Thanks for all the compliments regarding the newsletter. We shall try to "keep up the good work." But we do need the help we requested for handling some departments of the newsletter and functions of the User's Group written in PUG#6.

December 29, 1976
HERE AND THERE WITH PASCAL

(NEWS FROM MEMBERS, CONFERENCES, NEW BOOKS, APPLICATIONS PROGRAMS, ETC.)

is using Wirth's Algorithms + Data Structures = Programs and running lab problems on our batch system..." (*11/22/76*)

Flamih M. Oliver, 213 Weddel #12, Sunnyvale, CA 94086 (PUG member): "Has anyone implemented Pascal on an Interdata B/32? If so, who? I'm interested in the details." (*12/3/76*)

Ate Phung, Krefelder Str. 23, D-5100 Aachen, Germany (PUG member): "...As I am dealing with the implementation of Pascal to a Xerox Sigma 3, please send me any further informations about your experiences." (*12/8/76*)

Dean Schulz, INTEL Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051 (PUG member): "...please inform me of all implementations of Pascal for microprocessors of which you are aware." (*11/29/76*)

Stephen C. Schwarm, E.I. du Pont de Nemours Co., 101 Beech St., Wilmington, DE 19898 (PUG member): "...we now have 150 persons on the DECUS (DEC Users Group) SIG Pascal mailing list! I'm preparing a newsletter which I will send shortly, and forward a copy to you as well...there is a big demand for an RSX-11 compiler. ...P4 is great for implementing! The problems with PDP-11s is that the compiler uses too much data space: 5K for the stack and 16K dynamic storage. On the other hand P-code translates well to PDP 11 code: 48K and this can be reduced to 32K with some hand optimization...." (*1/3/77*)

Manfred Seifert, Informatik III, Universitat Karlsruhe, Zirkel 2, D-7500 Karlsruhe, Germany (PUG member): "Our institute is running a PDP 11/45 and two PDP 11/34s linked together via DL-11E and DL-11B. We are using RSX-11M and RSX-11S, communication with DEKNET-11 M/S. We are interested in a compiler under RSX-11M, or a compiler easily changeable to RSX support. We will use Pascal for programming IPC and decentralized control software in our local network." (*11/15/76*)

Rick Thomas, 408 Doner Ave., Takoma Park, MD 20012 (PUG member): "...we have a Univac 1108 installation at the University of Maryland and we just installed the Pascal/1100 compiler written at Naval Undersea Center by Mike Ball. We would be especially interested in an automatic Pascal source code indenter, for displaying the structure of someone else's code..." (*12/17/76*)

University of Washington, Seattle Computer Center Newsletter, August, 1976 shows that language processor statistics on its 6400/Cyber 73 system put Pascal at #2 behind Fortran (up from 5th the previous fiscal year). There was a decline in Simula usage over the same period. The Pascal increase was 65% from last year.

HERE AND THERE WITH PASCAL
(NEWS FROM MEMBERS, CONFERENCES, NEW BOOKS, APPLICATIONS PROGRAMS, ETC.)

CONFERENCES
The Third Annual Computer Studies Symposium at the University of Southampton being organized by D.W. Barron and J.M. Mullins has added a speaker in the Application part of the program. Olivier Lecarme will speak on "Pascal and Portability". The symposium, entitled: PASCAL, IMPLEMENTATION AND APPLICATIONS was fully described in Newsletter #6.

BOOKS AND ARTICLES
(* We really need someone to manage this section. *)

D.W. Barron reported in a letter dated 1 December, 1976 that there is a newly published book:


ROSTER 1/4/77 (NEW MEMBERS, CHANGED OR CORRECTED ADDRESSES AND PHONES)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address Details</th>
<th>City, State Zip Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>S. KAMAL ABDALI</td>
<td>ADD: A.D.R.</td>
<td>Cairo 12181, Egypt</td>
</tr>
<tr>
<td>GARY S. ANDERSON</td>
<td>ATTN: CENTRAL LIBRARY</td>
<td>38000 France</td>
</tr>
<tr>
<td></td>
<td>ATTN: COMPUTER CENTER USER SERVICES</td>
<td>98105</td>
</tr>
<tr>
<td></td>
<td>ATTN: COMPUTER UNIT</td>
<td>CV4 7AL UNITED KINGDOM</td>
</tr>
<tr>
<td></td>
<td>ATTN: THE LIBRARY</td>
<td>ISRAEL</td>
</tr>
<tr>
<td>HENRY R. BAUER III</td>
<td>82071</td>
<td></td>
</tr>
<tr>
<td>HERMAN BERG</td>
<td>53703</td>
<td></td>
</tr>
<tr>
<td>SCOTT BERTILSON</td>
<td>55495</td>
<td></td>
</tr>
<tr>
<td>JENI BEZIWIN</td>
<td>55301 FRANCE</td>
<td></td>
</tr>
<tr>
<td>I. N. BLAVINS</td>
<td>5006 AUSTRALIA</td>
<td></td>
</tr>
<tr>
<td>C. BRON</td>
<td>THE NETHERLANDS</td>
<td></td>
</tr>
<tr>
<td>J. A. CAMPELLE</td>
<td>2308 AUSTRALIA</td>
<td></td>
</tr>
<tr>
<td>JOHN CASEY</td>
<td>02115</td>
<td></td>
</tr>
<tr>
<td>JOHN E. COLLINS</td>
<td>55101</td>
<td></td>
</tr>
<tr>
<td>RICHARD CURE</td>
<td>94084</td>
<td></td>
</tr>
<tr>
<td>B. J. CORNELUS</td>
<td>H:6 7RK UNITED KINGDOM</td>
<td></td>
</tr>
<tr>
<td>LINDA E. COOLEY</td>
<td>94104</td>
<td></td>
</tr>
<tr>
<td>DENNIS DANCE</td>
<td>72204</td>
<td></td>
</tr>
<tr>
<td>DAVID DEMOREST</td>
<td>98124</td>
<td></td>
</tr>
<tr>
<td>PIERRE DESJARDINS</td>
<td>101 CANADA</td>
<td></td>
</tr>
<tr>
<td>FRED EILENSTEIN</td>
<td>81722</td>
<td></td>
</tr>
<tr>
<td>JACQUES FAIRE</td>
<td>75230 FRANCE</td>
<td></td>
</tr>
<tr>
<td>EDWARD E FERGUSON</td>
<td>76201</td>
<td></td>
</tr>
<tr>
<td>TED FISCHER</td>
<td>79222</td>
<td></td>
</tr>
<tr>
<td>DAVID C. FISZGELD</td>
<td>91740</td>
<td></td>
</tr>
<tr>
<td>J. J. FLORENTIN</td>
<td>WC2 7H United Kingdom</td>
<td></td>
</tr>
<tr>
<td>JUJ FONTANA</td>
<td>92704</td>
<td></td>
</tr>
<tr>
<td>DAVID N. GAY</td>
<td>78769</td>
<td></td>
</tr>
<tr>
<td>HILMAR GUSETLT</td>
<td>CH:3000 SWITZERLAND</td>
<td></td>
</tr>
<tr>
<td>C. C. HANDLEY</td>
<td>4000 SOUTH AFRICA</td>
<td></td>
</tr>
<tr>
<td>KAY A. HANSDOUREG</td>
<td>87544</td>
<td></td>
</tr>
<tr>
<td>KAY HARRISON</td>
<td>NZL 101 CANADA</td>
<td></td>
</tr>
<tr>
<td>KEVIN HAUSSMANN</td>
<td>55113</td>
<td></td>
</tr>
<tr>
<td>S. T. HEIDELBERG</td>
<td>94550</td>
<td></td>
</tr>
<tr>
<td>RICHARD HENDRICKSON</td>
<td>55420</td>
<td></td>
</tr>
<tr>
<td>TERUO HIKITA</td>
<td>D:6100 GERMANY</td>
<td></td>
</tr>
<tr>
<td>H.-J. HOFFMANN</td>
<td>35067</td>
<td></td>
</tr>
<tr>
<td>TIMOTHY J. HOFFMANN</td>
<td>55455</td>
<td></td>
</tr>
<tr>
<td>WILLIAM G. HOPKINS</td>
<td>17536</td>
<td></td>
</tr>
<tr>
<td>TAKAHIRO IMAIZUMI</td>
<td>53807</td>
<td></td>
</tr>
<tr>
<td>R. I. JOHNSON</td>
<td>58202</td>
<td></td>
</tr>
<tr>
<td>ED KATZ</td>
<td>70504</td>
<td></td>
</tr>
<tr>
<td>RICHARD K. KIEDRITZ</td>
<td>11760</td>
<td></td>
</tr>
<tr>
<td>JOHN A. LAIBERT</td>
<td>2300 AUSTRALIA</td>
<td></td>
</tr>
<tr>
<td>ROBERT M. LANSFORD</td>
<td>91107</td>
<td></td>
</tr>
<tr>
<td>ROBERT A. LAVER</td>
<td>55165</td>
<td></td>
</tr>
<tr>
<td>MIKE LEMON</td>
<td>61738</td>
<td></td>
</tr>
<tr>
<td>GEORGE LIGLER</td>
<td>75008</td>
<td></td>
</tr>
<tr>
<td>GARY LINDEGREN</td>
<td>84112</td>
<td></td>
</tr>
<tr>
<td>BRUCE LINK</td>
<td>87115</td>
<td></td>
</tr>
<tr>
<td>LEON NIJKASZEWICZ</td>
<td>00901 POLAND</td>
<td></td>
</tr>
<tr>
<td>ORLANDO S. MACGILLIE</td>
<td>95000</td>
<td></td>
</tr>
<tr>
<td>LARS MAGNUSON</td>
<td>S-751 21 SWEDEN</td>
<td></td>
</tr>
<tr>
<td>CRAIG MAJOLIN</td>
<td>92037</td>
<td></td>
</tr>
<tr>
<td>KONRAD WIENER</td>
<td>A-1150 AUSTRIA</td>
<td></td>
</tr>
<tr>
<td>'THOMAS G. MOIGETY</td>
<td>02053</td>
<td></td>
</tr>
<tr>
<td>BRIAN MEBK</td>
<td>W:7AH UNITED KINGDOM</td>
<td></td>
</tr>
<tr>
<td>B. T. MITCHELL</td>
<td>KY16 UNITED KINGDOM</td>
<td></td>
</tr>
<tr>
<td>RICHARD B. MEBK</td>
<td>Dept. of Computer Sci.</td>
<td></td>
</tr>
<tr>
<td>SUNY AT STONY BROOK</td>
<td>STONY BROOK NY 11794</td>
<td></td>
</tr>
<tr>
<td>(516) 246-5937</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROY A. WILSKER</td>
<td>27 BENNETT STREET</td>
<td></td>
</tr>
<tr>
<td>BOSTON MA 02154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(617) 899-6638</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RICHARD B. MEBK</td>
<td>Dept. of Computer Sci.</td>
<td></td>
</tr>
<tr>
<td>SUNY AT STONY BROOK</td>
<td>STONY BROOK NY 11794</td>
<td></td>
</tr>
<tr>
<td>(516) 246-5937</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ARTICLES
(FORMAL SUBMITTED CONTRIBUTIONS)

LIFE, LIBERTY AND THE PURSUIT OF UNFORMATTED INPUT
D.W. BARRON AND J.M. MULLINS
University of Southampton

In PUGN#5, Eisenberg presents three examples which, he claims, demonstrate the necessity for formatted input. This note attempts to demolish those claims.

Example 1 is concerned with survey analysis. 75 candidates are rated on a scale 1 to 5, and the observations from each individual taking part in the survey are punched as a contiguous stream of 75 digits. Formatted input allows this to be read as 75 integers; extracting the integer values by reading characters and using ord(ch) - ord('0') is said to "detract from the exercise ... the student is not interested in the use of "ord" or strings...."

The student needs to realise that dealing with this sort of data is messy, so that in future he can tell people designing surveys to prepare their data in a more palatable manner, instead of producing unreadable and inherently error prone sequences of digits. (Anyone designing a survey ought to consult the person responsible for analysing the data before the data is punched. They can then be told that redundancy in the form of separators allows error checking.)

Example 2 "Your Ph.D. Advisor" has produced a card deck in which "columns 11 to 70 contain 20 3-digit numbers...".

See the remarks on example 1 above. If your Ph.D. Advisor really produces data like this, he ought not to be allowed to advise Computer Science Ph.D. students. We are asked to note that omitting blanks "saves keypunching time and storage space, both of which are the equivalent of money". Balderdash. Do the sums, and compare the cost with the time of the (expensive) programmer sorting out the resultant mess. (Incidentally, anyone who tells us to keypunch real numbers in a contiguous stream to avoid "wasting time typing spaces" will be told, politely but firmly, to go get his head examined.)

Example 3 is presented as a situation where formatted input "is a near necessity". The problem is to extract the three fields from the 7-digit code on a pair of jeans, where the first three digits code the style, the next two code the color, and the last two digits code the material. Who needs formatted input? How about:

```pascal
var code, style, color, material : integer;
read(code);
material := code mod 100;
color := (code div 100) mod 100;
styLe := code div 10000;
```

This method works as long as we can read the largest code, 7070436, which means we need at least 24-bit arithmetic. Doing it this way in a class has the advantage that the student will understand mod and div at the end of it, as well as learning something about positional coding systems, the representation of integers, and the significance of maxint.

If we want to avoid the maxint restriction, we have to use the dreaded 'ord'. But is it so terrible? If we have defined type digit = 0..9 and we are prepared to take the accuracy of the input on trust (we shouldn't, but formatted input schemes do), all we need is

```pascal
function nextdigit : digit;
var ch : char;
begin
read(ch);
nextdigit := ord(ch) - ord('0')
end;
```

If the student is dealing with these kind of codes it will do him no harm to use ord, since in any case he ought to appreciate the difference between the character '3' and the number 3.
Finally, if we want to be fancy we can define

```
type field = (onecolumn, twocolumns, threecolumns);
and
function formattedread (f : field); integer;
begin
  case f of
    onecolumn : formattedread := nextdigit;
    twocolumns : formattedread := 10*nextdigit + nextdigit;
    threecolumns : formattedread := 100*nextdigit + 10*nextdigit + nextdigit
  end;
end;
```

OK, this only does what formatted input would have done. But we can amend nextdigit to catch input errors, and educationally it is far better to learn to do it this way than it is to learn how to construct format statements.

There is only one moral to be drawn from the defence of formatted input:

**WARNING: FORTRAN CAN IMPAIR YOUR JUDGEMENT**

(*Received 11/29/76*)
B. Using the PRINTER PLOTTER:

The PASCAL PRINTER PLOTTER is easy to use. Only a few declarations are necessary. Four separate procedures must be considered: SCLINITIAL, SCLWINDOW, SCLPLOT, and SCLPRINT.

Simple plots are made by calling the basic procedures once, first SCLINITIAL, then SCLPLOT, and finally SCLPRINT. Multiplots are made by repetitively calling SCLPLOT. The first call establishes the scale, and all further calls will then be handled according to that scale. Any points lying outside the boundaries established by either SCLPLOT (first call), or a call to SCLWINDOW will be ignored.

SCLWINDOW acts as if it were the first call to SCLPLOT except there is no plotting. SCLWINDOW provides the expansion facility by allowing the user to specify ranges of X and Y values directly.

B.1 Variable and Type Declarations:

Certain declarations are required by the PASCAL PRINTER PLOTTER. The user should insert them into his MAIN PROGRAM. The most important is an ARRAY of 700 words which SCLPLOT uses as a large work area. Since the four procedures also communicate through this ARRAY, it is very important that it not be altered by the user. Identifiers used in the following illustration are only suggested. In other words, the large work area, which will be referred to as IMAGE, could just as well be called XYZ.

Example B.1

PROGRAM TESTPLOTTER(INPUT, OUTPUT);
CONST;
NUM = 50; (*MAX NUMBER OF DATA POINTS TO BE PLOTTED*)
TYPE
SCRATCH = ARRAY [1..700] OF INTEGER;
ARDATA = ARRAY [1..NUM] OF REAL;
CH60 = PACKED ARRAY [1..60] OF CHAR;
VAR
X: ARDATA; (*ARRAY OF X-VALUES*)
Y: ARDATA; (*ARRAY OF CORRESPONDING Y-VALUES*)
N: INTEGER; (*NUMBER OF DATA POINTS TO BE PLOTTED*)
IMAGE : SCRATCH; (*WORK AREA FOR PASCAL PRINTER PLOTTER*)
TITLE : CH60; (*TITLE OF THE PLOT *)
XLEG : CH60; (*HORIZONTAL HEADING FOR THE X-AXIS*)
YLEG : CH60; (*VERTICAL HEADING FOR THE Y-AXIS*)
CH: CHAR; (*PLOTTING CHARACTER*)

Note:

It is important that SCRATCH, ARDATA, and CH60, or any other names used for these, be defined exactly as shown above. SCLINITIAL, SCLWINDOW, SCLPLOT, and SCLPRINT expect this and cannot function otherwise. Also, the variables: TITLE, XLEG, YLEG, CH, and N are optional. This depends upon the particular form of PROCEDURE usage. See section B.1 and example B.1.3.

B.2 Procedure Declarations:

The user should also declare the four procedures: SCLINITIAL, SCLWINDOW, SCLPLOT, and SCLPRINT in the MAIN PROGRAM. Notice that IMAGE, PROFILE, X, and Y are the only variable parameters. X1, X2, Y1, Y2, N, CH, TITLE, XLEG, and YLEG are value parameters. Value parameters can be specified as constants, directly in a PROCEDURE call. This can help eliminate extra variables and also can be an aid to readability.

PROCEDURE declarations must appear as shown below. Again, identifiers can be anything convenient, however positioning is extremely important. For example, in SCLPLOT, the parameters: IMAGE , N , X , Y , CH must appear in that order. X , Y , N , CH , IMAGE is syntactically incorrect.

Example B.2

PROCEDURE SCLINITIAL(VAR IMAGE: SCRATCH); EXTERN;
PROCEDURE SCLWINDOW(VAR IMAGE: SCRATCH;
X1, X2, Y1, Y2: REAL); EXTERN;
PROCEDURE SCLPLOT(VAR IMAGE: SCRATCH;
N: INTEGER;
VAR X, Y: ARDATA;
CH: CHAR); EXTERN;
PROCEDURE SCLPRINT(VAR IMAGE: SCRATCH; VAR PROFILE: TEXT;
TITLE, XLEG, YLEG, CH60); EXTERN;

Note:

PROCEDURE declarations must be made under the default compiler X option, that is, X4 for PASCAL 6000.
B.3. Typical PROCEDURE Usage:

The following are four examples of typical PROCEDURE usage. The first three are single plots. The fourth is a multiplot. The second is an expansion of the first using SCLWINDOW. The third shows how convenient it can be to substitute constants directly as ACTUAL PARAMETERS, thus eliminating extra variables. Notice that TITLE, XLEG, and YLEG are actually strings. Strings are defined in PASCAL as PACKED ARRAYS OF CHAR. Also, note that SCLPRINT writes the graph to the file, OUTPUT. A different external file, other than OUTPUT, can be used if desired.

Example B.3.1

SCLINITIAL(IMAGE); (* INITIALIZE *)
SCLPLOT(IMAGE,N,X,Y,CH); (* PLOT *)
SCLPRINT(IMAGE,OUTPUT,TITLE,XLEG,YLEG); (* PRINT *)

Example B.3.2

SCLINITIAL(IMAGE); (* INITIALIZE *)
SCLPLOT(IMAGE,N,X,Y,CH); (* PLOT *)
SCLPRINT(IMAGE,OUTPUT,TITLE,XLEG,YLEG); (* PRINT *)

Example B.3.3

SCLINITIAL(IMAGE); (* INITIALIZE *)
SCLPLOT(IMAGE,47,X,Y,"P"); (* ALL FURTHER REFERENCES TO SCLPLOT ARE DEPENDENT UPON THE SCALE JUST ESTABLISHED. *)
SCLPRINT(IMAGE,OUTPUT,
".................. THIS IS THE TITLE .................."); (* CONSTANTS ARE SUBSTITUTED FOR TITLE, XLEG, AND YLEG *)

Example B.3.4

SCLINITIAL(IMAGE); (* ESTABLISH SCALING *)
SCLPLOT(IMAGE,N,X,Y,"A"); (* ALL FURTHER REFERENCES TO SCLPLOT ARE DEPENDENT UPON THE SCALE JUST ESTABLISHED. *)
SCLPLOT(IMAGE,N,X,Y,"B");
SCLPRINT(IMAGE,OUTPUT,TITLE,XLEG,YLEG); (*PRINTS THE MULTIPLOT*)

C. Programming Examples:

Three different programming examples will be given at the end of this writeup. The first is an example of a single plot. It shows what happens if the data is too large or small to have properly displayed axis labels. A scale factor message is printed beside each axis affected. The second is a multiplot. Finally, the third is an expansion of the second showing how convenient it can be to substitute constants directly as ACTUAL PARAMETERS, thus eliminating extra variables. Notice that TITLE, XLEG, and YLEG are actually strings. Strings are defined in PASCAL as PACKED ARRAYS OF CHAR. Also, note that SCLPRINT writes the graph to the file, OUTPUT. A different external file, other than OUTPUT, can be used if desired.

Example B.3.1

SCLINITIAL(IMAGE); (* INITIALIZE *)
SCLPLOT(IMAGE,N,X,Y,CH); (* PLOT *)
SCLPRINT(IMAGE,OUTPUT,TITLE,XLEG,YLEG); (* PRINT *)

Example B.3.2

SCLINITIAL(IMAGE); (* INITIALIZE *)
SCLPLOT(IMAGE,N,X,Y,CH); (* PLOT *)
SCLPRINT(IMAGE,OUTPUT,TITLE,XLEG,YLEG); (* PRINT *)

Example B.3.3

SCLINITIAL(IMAGE); (* INITIALIZE *)
SCLPLOT(IMAGE,N,X,Y,"P"); (* ALL FURTHER REFERENCES TO SCLPLOT ARE DEPENDENT UPON THE SCALE JUST ESTABLISHED. *)
SCLPRINT(IMAGE,OUTPUT,
".................. THIS IS THE TITLE .................."); (* CONSTANTS ARE SUBSTITUTED FOR TITLE, XLEG, AND YLEG *)

Example B.3.4

SCLINITIAL(IMAGE); (* ESTABLISH SCALING *)
SCLPLOT(IMAGE,N,X,Y,"A"); (* ALL FURTHER REFERENCES TO SCLPLOT ARE DEPENDENT UPON THE SCALE JUST ESTABLISHED. *)
SCLPLOT(IMAGE,N,X,Y,"B");
SCLPRINT(IMAGE,OUTPUT,TITLE,XLEG,YLEG); (*PRINTS THE MULTIPLOT*)

D. Hints, Cautions, Errors:

There is one mistake a user could make which can cause unpredictable results, that is, failure to call SCLINITIAL.

The usual syntax precautions apply here as well as in any PASCAL program. Make sure that the parameters are of the right TYPE and see that they are properly positioned. Do not forget that the variable parameters are: IMAGE, PROFILE, X, and Y, and that the value parameters are: Xi, Yi, Y2, X2, CH, TITLE, XLEG, and YLEG. Also, if the user is manipulating compiler options, the X option must be set to X4 for PROCEDURE declarations.
If for each call to SCLPLOT, N is less than or equal to zero, and either SCLWINDOW is called with the X-values or the Y-values being equal, or is not called at all, then when SCLPRINT is called a message will be printed: "EMPTY GRAPH".

Normally, when calling SCLPRINT, the OUTPUT file is used. See - Typical Procedure Usage. However, a different external file could just as well be used provided that it is properly declared in the number of X and Y values and also if N is zero, and either without equal to zero, everything will work properly. It should go approximately graph scales which centers the information in the graphical field specifications. The plot image of numerically printed.

In reference to SCLPLOT, as long as N agrees with the number of X and Y values and also if N is not less than or equal to zero, everything will work properly. It should go without saying that any N passed to SCLPLOT should lie within the range of INTEGER numbers as defined by PASCAL 6000.

E. Notes:

The external routines described herein require approximately 2300 octal words of central memory.


This reference gives an excellent method for choosing graph scales which centers the information in the graphical frame and provides divisions which are simple numbers. A few changes have been made to handle special cases (e.g., when all values are on a single X or Y line) and to re-scale to fit PASCAL field specifications. The plot image of numerically rounded points is built up in the scratch array and is then printed.

J5 MINN SCLPLOT, by M. Frisch, revised February 1971, describing a FORTRAN printer plotting routine, was also used as a reference.

***************************************************************
A Copy of This Writeup Can be Obtained in Two Ways:
Check room 140 Experimental Engineering.
Look for: -PASCAL PRINTER PLOTTER-
Use the CYBER 74 or 6400 machine.
Execute the control statement: WRITEUP(PASCLIB=PRNTPLT)
***************************************************************
(*Received 12/4/76*)
Y SCALE FACTOR: 10** -3.

THIS PLOT SHOWS EXTREMES.

Y "0"
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
Y "*
A: \( Y = \sin(x) \);  \( R = \cos(x) \);  \( C: y = x/0.4 - 0.8 \);  \( D: y = 0.5 \)

![Diagram](image_url)
A: Y=SIN(X);  B: Y=COS(X);  C: Y=X/0.4-0.8;  D: Y=0.5

FIGURE C.3

X INCREASING

PASCAL NEWSLETTER #7
FEBRUARY, 1977

PAGE 16
In the 1974-1975 school year the Computer Science Department at the University of Minnesota decided that more extensive use of PASCAL would be made in their programming language courses. Since much of this use would be interactive, one concern of the University Computer Center was that PASCAL required the most memory for compilation (approx. 55000 bytes) of the interactive languages available (for comparison note that BASIC and APL/CYBER require approximately 25000 bytes and MNF FORTRAN 44000 bytes) on our Instructional Time Sharing CDC 6400. Several years experience in running a large volume (202 maximum simultaneous users and 350,000 runs in April 1975 see reference [5]) time sharing service had shown that a successful time sharing service on CDC 6000 machines required that each user be limited to at most 54000 bytes of memory. (In addition to a restriction on memory for each user there are requirements for enough mass storage channels and peripheral processors ... but that is another story.) Since the PASCAL level 9 compiler required 421218 bytes to load and in addition allocated buffers, "stack" and "heap" space, I decided that a reduction in PASCAL load size could be accomplished by rethinking certain PASCAL code generations in the area of procedure calls, constant loads, case statement jumps and/or by combining common procedures (+, , divide, mod, in-line functions). In addition when Urs Ammann of ETH, Zurich was informed of the project, he suggested several core reduction ideas that were implemented by John Strait of our staff in late 1975. In a compiler-compiler the generation of faster and shorter code helps not only the user, but also since PASCAL 6000 compiles itself this will be reflected in a smaller and faster compiler.

In trying to analyze where core reductions can be made in a compiler such as PASCAL, it is worthwhile obtaining several tables that help the core reducer to concentrate on the essentials of the language (or program) that are amenable to reduction techniques. The first such table is the count of the static number of procedure calls ordered in descending use.

In PASCAL 6000 with a load length of approximately 18,000-60 bit words (431308) there are approximately 2300 static (meaning physically present) procedure calls distributed among the approximately 140 procedures that constitute the compiler. The top 10% of the called procedures are listed in the following table with an additional break down into three main areas: code generation, symbol input and error message.

<table>
<thead>
<tr>
<th>TABLE 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>ERROR</td>
</tr>
<tr>
<td>GEN15</td>
</tr>
<tr>
<td>INSYMBOL</td>
</tr>
<tr>
<td>GEN30</td>
</tr>
<tr>
<td>COMPTYPES</td>
</tr>
<tr>
<td>DECREFX</td>
</tr>
<tr>
<td>NEEDX</td>
</tr>
<tr>
<td>LOAD</td>
</tr>
<tr>
<td>SKIP</td>
</tr>
<tr>
<td>NOOP</td>
</tr>
<tr>
<td>CLEARREGS</td>
</tr>
<tr>
<td>EXPRESSION</td>
</tr>
<tr>
<td>NEXTCH</td>
</tr>
<tr>
<td>OPERATION</td>
</tr>
<tr>
<td>TOTAL</td>
</tr>
</tbody>
</table>

% of 2300 calls: 46%
% of load core: 21%

In trying to analyze where core reductions can be made in a compiler such as PASCAL, it is worthwhile obtaining several tables that help the

<table>
<thead>
<tr>
<th>CODE GENERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>GEN15</td>
</tr>
<tr>
<td>GEN30</td>
</tr>
<tr>
<td>COMPTYPES</td>
</tr>
<tr>
<td>DECREFX</td>
</tr>
<tr>
<td>NEEDX</td>
</tr>
<tr>
<td>LOAD</td>
</tr>
<tr>
<td>NOOP</td>
</tr>
<tr>
<td>CLEARREGS</td>
</tr>
<tr>
<td>OPERATION</td>
</tr>
<tr>
<td>TOTALS</td>
</tr>
<tr>
<td>1055</td>
</tr>
</tbody>
</table>

% of 2300 calls: 46%
% of load core: 21%

In trying to analyze where core reductions can be made in a compiler such as PASCAL, it is worthwhile obtaining several tables that help the
Except for EXPRESSION these procedures fall into the three groups mentioned above and account for 76% of the static procedure calls and 25.5 per cent of the total loaded length of PASCAL 6000. Thus, a large savings can be obtained for each word saved in a static procedure call (2300 ≈ n words saved).

Another place to look for central memory reductions is in the prologue and exit of each procedure. Since there are 140 procedures in the PASCAL 6000, each word saved represents 140 ≈ n (214g) additional cells saved. Candidates for this reduction are the procedure name word, the word containing the lengths of the executable and total procedure code, the three words for procedure initialization, and the 1 1/2 words for procedure exit. The first two are needed for the current POST MORTEM dump, but could be eliminated if the POST MORTEM dump would obtain these words from the LGO file or would build a specific POST MORTEM file at load time for possible use when errors occurred. The procedure initialization and exit can be reduced to one word and three quarters respectively by use of common entry and exit routines at a probable 8% slow down in compilation speed due to the slowness of NEXTCH (actually 25% of compilation time on a CYBER 74 is currently spent in NEXTCH, and by improving this routine the 8% compilation slow down can be more than compensated for by the NEXTCH speed up).

One final way to look for central memory reduction in the compiler is to look at a table of the largest PASCAL 6000 procedures. This uses the theory that it always pays to look at the fattest routines. When looking at this table the core reducer should look for similar functionality that can be combined into one routine or broken out into a simple subroutine. In addition the longest routines are examined for exactly what makes them long, with a look to improving code production that can reduce the length (in PASCAL it may be the nesting depth of procedure calls from that routine, a poorly done CASE statement, or a rethinking of code generation). Finally, the reverse of top down step wise refinement (integration) can sometimes achieve good savings as was obtained by combining the procedures ROUNDF, ABSF, SQRF, TRUNFC, ODDP, ORDF, CHRF, PREDSUCCF, CARDF, EXPDF, into the routine STDINLINEFUNCS.

<table>
<thead>
<tr>
<th>Table 2</th>
<th>PASCAL 6000 procedures ordered by length in PASCAL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler Procedure Name</td>
<td>Length PASCAL 2</td>
</tr>
<tr>
<td>CALLNONSTANDARD</td>
<td>1266g</td>
</tr>
<tr>
<td>STORE</td>
<td>1244g</td>
</tr>
<tr>
<td>LOAD</td>
<td>1200g</td>
</tr>
<tr>
<td>TERM</td>
<td>1027g</td>
</tr>
<tr>
<td>EXPRESSION</td>
<td>1025g</td>
</tr>
<tr>
<td>FACTOR</td>
<td>1006g</td>
</tr>
<tr>
<td>BODY</td>
<td>754g</td>
</tr>
<tr>
<td>TYP</td>
<td>712g</td>
</tr>
<tr>
<td>INSYMBOL</td>
<td>655g</td>
</tr>
<tr>
<td>WRITE</td>
<td>603g</td>
</tr>
<tr>
<td>UNPACK</td>
<td>535g</td>
</tr>
<tr>
<td>FORSTATEMENT</td>
<td>523g</td>
</tr>
<tr>
<td>FIELDLIST</td>
<td>515g</td>
</tr>
<tr>
<td>INDEXCODE</td>
<td>504g</td>
</tr>
<tr>
<td>CASESTATEMENT</td>
<td>503g</td>
</tr>
<tr>
<td>PACK</td>
<td>473g</td>
</tr>
<tr>
<td>PARAMETERLIST</td>
<td>472g</td>
</tr>
<tr>
<td>SIMPLEXPRESSION</td>
<td>461g</td>
</tr>
<tr>
<td>PROCEDUREDECLARATION</td>
<td>427g</td>
</tr>
<tr>
<td>STDINLINEFUNCS [as separate routines]</td>
<td>[1350g]</td>
</tr>
</tbody>
</table>

Not all of the code reduction features were viable or even desirable; but since the application of each reduction was applied to the previous one, the following table briefly describes the change, the octal (decimal) savings
Change 0
Stock level 9 compiler

Change 1
Avoid extra SBI Xj commands when loading a base address

Change 2
Avoid extra Bxi Xj commands at procedure calls

Change 3
Use 2 jump commands/word in CASE statements

Change 4
Correct inefficient case statement in the procedure "STATEMENT"

Change 5
Use common ENTRY/EXIT routines

Change 6
Make the "RJTOEXT" procedure reasonable

Change 7
Use a single procedure for the inline functions (GDO...CARD) called STDIINLIN FUNC

Change 8
Eliminate extra stack manipulation

Change 9
Use common ENTRY/EXIT routines

Change 10
Make the "RJTOEXT" procedure reasonable

Change 11
Use common code for "++" and "+-"

Change 12
Use S5 = MX5 number-1 of static indirects to load if static link not available

Change 13
Use RJ procedure rather than S5X return, JP procedure

Examples of the code generated in the application of these principles are given in the following pages for changes 3, 5, 9, 10, 12 and 13. Note also that the application of change 13 disallows 10.

Change 3
Example of the CASE statement
From procedure OPTIONS

Pascal

<table>
<thead>
<tr>
<th>CASE</th>
<th>CHI</th>
<th>IF</th>
<th>OF</th>
<th>SAI</th>
<th>CHI</th>
<th>SB3</th>
<th>x1</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>B</td>
<td>.16</td>
<td>B</td>
<td>SAI</td>
<td>B3+113</td>
<td>SB3</td>
<td>x1</td>
</tr>
<tr>
<td>20</td>
<td>'B'</td>
<td>.17</td>
<td>JP</td>
<td>B3+113</td>
<td>SB3</td>
<td>x1</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>'E'.20</td>
<td>(code for case 'B' +)</td>
<td>'E'.27</td>
<td>(code for case 'E', etc. +)</td>
<td>'P'</td>
<td>.113</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'T'</td>
<td>.114</td>
<td>JP</td>
<td>.144</td>
<td>(etc.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>'U'</td>
<td>.115</td>
<td>JP</td>
<td>.20</td>
<td>(etc.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>'X'</td>
<td>.116</td>
<td>JP</td>
<td>.144</td>
<td>(etc.)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
is changed to 1 word
SA7 needed
RJ P.ENTRB
the current exit code of
SA1 BS+BI
SB6 BS
SB7 X1
LXI 42
SB5 XI
JP B7
1 3/4 words
SB7 X1
LXI 42
SB6 BS
SB5 XI
JP B7
Is changed to 3/4 word
SA1 BS+BI
JP P.EXIT
where P.EXIT
SB7 XI
LXI 42
SB6 BS
SB5 XI
JP B7
is changed to 1 word
SA7 needed
RJ P.ENTRB
(see code for 12 and 13)
reset NEXT
set return address
restore old STACK BASE

For a CYBER 74 the change in P.EXIT ratio in cycle speed is 33/26 = 1.27;
for the 6400 the ratio is 63/51 = 1.24.

Change 9
Examples of the MXi and LXi commands for constants.

In PASCAL the very useful powerset often uses constants that have
contiguous bit sequences. For the CDC 6000 machines it is usually best
to have code productions that are two 15-bit commands rather than a single
30-bit command, since the 30-bit command often will not fit into the current
word causing a non-useful NO command to be produced. The MASK and SHIFT
commands of the CDC 6000 allow any constant of the form \((2^n-1)2^m\) to be
generated by MXi n; LXi m+n. If the constant value is greater than 217
this is always the best method since it takes (on a 6400) the same or less
time and saves the 60-bit word holding that constant. If the constant is
less than 217 and there is only 15-bits left in the current word, this method
will avoid the useless NO command.

EXAMPLES

a) PROCEDURE WITHSTATEMENT (COMP 6147, 6148)
IF CBDFSPL <> 0 THEN
    BEGIN NEEDX(0,7,1)
    CURRENT PASCAL
    ZR X7,,143 BXO X6 NO
    SXI 7 SX2 BS+20
    SX6 BS SX7 .120 NO
    EQ NEEDX
b) PROCEDURE OPTIONS (COMP 407)
'B' IF CH IN ['1'..'9']

MODIFIED PASCAL

LXI 3 SX2 BS+20 MX5 1
RJ NEEDX

MODIFIED PASCAL

CBUF = CBUF=1000B+1000B+777777B+FK
CURRENT BX5 X6 LX5 9 LX5 9 NO
SA1 217
CBUF = CBUF=1000000B+777777B+FK
MODIFIED BX5 X6 LX5 18 MX1 18 LX1 18

Change 10
Example from Procedure LOAD (COMP 2792-2796)

PASCAL

CODE GENERATION
IF SVAL = 0 then GEN15(13B,1,1,1)
125 IF SVAL = 1 then GEN15(76B,1,1,0)
131 IF SVAL = 2 then GEN15(76B,1,1,1)
ELSE
    125 JP GEN15
    124 JP 141
    125 BX4 X2-XO NZ X4,,131 NO
    126 SXO 768 SX2 B1 SX6 B7
    127 SX7 ,130 JP GEN15
    130 EQ ,141
    131 SX4 B1+B1 IX5 X2-X4 NZ X5,,136
    132 SXO 768 SX2 B1 BX3 X2
    133 SX6 B7 SX7 ,135
    134 JP GEN15
    135 JP ,141
    136

Rule 1
If the termination of a THEN clause is a procedure call and there is no
other clause ending at the same point before an ELSE; more efficient code
can be generated by eliminating the standard JP to terminal IF point by
making the return address of that procedure call go to the terminal IF point.
The above example code would produce the following, saving three words and
the corresponding jump times:
Rule 2

If the termination of an individual CASE is a procedure call and there is no other clause ending at the same point (i.e., more than one address that refers to such a point) then the JP to the terminal CASE point can be eliminated by making the return address and that procedure call go the terminal case point. The coding produced is similar to that in the THEN-ELSE.

Changes 12 and 13

Example from the procedure FACTOR.

Note that this modification depended on Change 5, the use of a common ENTRY point routine.

Previous CODE GENERATED FOR DECREFX(I) and FACTOR(FSYS):

<table>
<thead>
<tr>
<th>DECREFX(I)</th>
<th>FACTOR(FSYS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP 5200</td>
<td>COMP 5115</td>
</tr>
<tr>
<td>SA1 B5+12</td>
<td>SA1 B5+3</td>
</tr>
<tr>
<td>BX0 X1</td>
<td>BX0 X1</td>
</tr>
</tbody>
</table>

3 ½ words

<table>
<thead>
<tr>
<th>SA2 B5</th>
<th>follow 2 ½ words</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA2 X2</td>
<td>static</td>
</tr>
<tr>
<td>SA2 X2</td>
<td>link</td>
</tr>
<tr>
<td>BX6 X2</td>
<td></td>
</tr>
</tbody>
</table>

5 words

<table>
<thead>
<tr>
<th>SX7 =+2</th>
<th>set return address</th>
</tr>
</thead>
<tbody>
<tr>
<td>SX6 X6</td>
<td>Eq DECREFX transfer to routine</td>
</tr>
</tbody>
</table>

Using changes 12 and 13:

<table>
<thead>
<tr>
<th>SA1 B5+12</th>
<th>1 ½ words</th>
</tr>
</thead>
<tbody>
<tr>
<td>BX0 X1</td>
<td></td>
</tr>
</tbody>
</table>

1 ½ words

<table>
<thead>
<tr>
<th>MX5 h</th>
<th>if base not in B register</th>
</tr>
</thead>
<tbody>
<tr>
<td>RJ DECREFX</td>
<td>transfer to routine</td>
</tr>
</tbody>
</table>

P.ENTRB

Is an example of the common ENTRY routine where the static link is stored on the stack and a RJ ROUTINE is used rather than the SX7 RETURN JP ROUTINE.

X5 = STATIC LINK (IF X5 > 0) else MX5 LEVEL-PFELEV-1

where LEVEL and PFELEV are the LEVELS of the caller and called procedure respectively.

P.ENTRB DATA 0

<table>
<thead>
<tr>
<th>PL X5, P.ENT2</th>
</tr>
</thead>
<tbody>
<tr>
<td>BX6 X5</td>
</tr>
<tr>
<td>SA5 B5</td>
</tr>
<tr>
<td>SA5 X5</td>
</tr>
<tr>
<td>LX6 1</td>
</tr>
<tr>
<td>NG X6, ^</td>
</tr>
</tbody>
</table>

P.ENT2 BX6 X5 store static link on stack

<table>
<thead>
<tr>
<th>SA5 P.ENTRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX5 30</td>
</tr>
<tr>
<td>SA5 X5=2</td>
</tr>
<tr>
<td>AX5 30</td>
</tr>
<tr>
<td>SX7 X5</td>
</tr>
</tbody>
</table>

| BX6 B5 |
| LX6 18 |
| SX7 X7X6 |
| SB5 B6 |
| SA7 B6+B1 |
| SB6 B6+87 |
| SB7 B6+100 |
| LT B7,B4,P.ENTRB |

This change was the most controversial since although it gave dramatic reduction in central memory, it changed the manner of procedure calls from that documented in reference [3] and definitely slowed down compilation by 8% on the CYBER 74. One way to use the core reduction and slower procedure linkage would be to have the PASCAL compiler internally use the RETURN JUMP method of procedure calls while generating the forms SXI return and JP procedure for user execution binary. This would mean that three assemblies of PASCAL would be required to update to a new PASCAL rather than the two assemblies required presently.
Comparisons

Some comparisons with PASCAL 1 described in [1] are interesting.

(1) by instruction length

1971 PASCAL 1
15,925 48.7% long instructions
9,385 28.2% short instructions (15-bit)
7,456 22.8% padding instruction (NOOP)

15,925 39,645 words
12,173 39,645 words

(2) by instruction type

27.6% fetch and store
15.0% logical and shift and mask
3.2% arithmetic
14.4% base address register
10.5% load literal
22.8% (NOOPS)

7,010 19,676 13,151 100% = 13,151 100% = 13,151 words
7,456 21.5% long instructions
12,173 28.7% short instructions
14.4% logical and shift and mask
6.2% base address register
10.5% load literal
22.8% (NOOPS)

Reference [3] shows that register remembrances, BX, X6,7 rather than
NOOP's and more efficient code generation were one aim in the design of
PASCAL2. This manifests in PASCAL2 in increased subroutine calls (from
10.5% to 12.4%) to produce the more efficient code and in the reduction of
fetch and store commands (from 27.6% to 21.5%). In addition to [3]'s
replacement of NOOP's with logical commands the generation of M5S level; RJ
PROCEDURE compared with PASCAL's SX7 RETURN EQ PROCEDURE caused increased
mask commands and a corresponding decrease in LOAD LITERALs (SKI value).

Note that in PASCAL1 14.3% of the loaded space (NOOP commands) is not
used compared with 13.3% in the modified PASCAL. The designer of the CDC
6000 machines has recognized this loss in his latest machine the CRAY 1.
In that machine JUMPS are to any 16-bit portion of the 64-bit word rather
than to the top most portion of the word as is done in a CDC 6000
machine. Thus the 4,446 non executable NOOPS (of 7010 total) could be elimi-
nated if the CDC 6000 machines had such a feature allowing 8.5% of the loaded
compile space to be saved.

Our last way to shorten the compiler is to rewrite the code generation
part which currently comprises about one-fourth of the total length.
The current code generation scheme seems to have two main deficien-
cies. The target computer code that PASCAL generates is spread throughout rather
than gathered in functional groupings in the compiler. Second, for

Pseudo register of ATTR:
I=0; J=1; K=2; L=3; M=4; N=5; O=6; P=7
Thus approximately 29 cells are replaced by 6 cells which compares favorably with that reduction obtained in the MNF compiler where 1900 pseudo commands occupied 709 words and required a MACRO interpreter of 350 words. If the pseudo commands of MNF were done as normal procedure calls it would have taken approximately 1900 \times 2 \frac{3}{4} = 5225 words in the MNF compiler. Thus the total saving is 4000 words or an 80% reduction in that code generation portion of the MNF compiler. Applying this to the PASCAL compiler with approximately 4000 words for 1055 code generation procedures calls and assuming a MACRO interpreter of 500 words and 100 calls to the MACRO interpreter we can estimate a 2500-3000 word reduction in the loaded length of the PASCAL compiler.

Note that the implementation of this assumes a VALUE declaration initialization of packed records which is currently not available in PASCAL. The actual implementation of the MACRO skeleton is not as simple as my example but reference [4] or the MNF compiler listing give additional pseudo commands and implementation techniques.

Conclusions

PASCAL 2 is amenable to several different methods of compiler length reduction. As a fellow compiler writer (although since MNF is written in machine language it may be compared with the last of the dinosaurs speaking to Homo sapiens), I would rather see the full language specification and one standard compiler, than to see small subsets such as PASCAL-5. For this reason I think it essential to improve PASCAL2 and with the reductions discussed in this article it should be possible to obtain load lengths of approximately 30\,\text{k} for the full language rather than the current 44\,\text{k} on a CDC 6000 (i.e. a reduction by one-third).

References


(*Received 12/29/76*)
Dear Andy,

Reading "A Primer on PASCAL" by Conway Gries and Zimmerman, my eye was caught by the fact that every time they print a line in their programs they start it with an explicit blank. After doing this without explanation for many pages, they eventually explain about carriage control characters, a la FORTRAN. Unfortunately, they have some justification from §14 of the REPORT: "...The first character on each line of printfiles may be interpreted as a printer control character..."

To my mind it is deplorable that a clean language should be soiled by such a hangover from the bad old days. In most of the "primitive" languages, I/O is relegated to a sub-language of a wholly ad hoc nature, e.g. FORTRAN format statements and carriage control characters. In PASCAL we have got away from this: we have "writeln" and "page", and if the system has to put a '1' in front of the print line to suit the peculiarities of the line printer, that is the system's affair. I don't want to put it there myself, because my terminal will actually print the '1' - it expects ASCII code FF to signal a page feed. If everybody agrees that "page" means "move to head of form", our programs are more likely to be transportable, so why the special provision for carriage control characters in the Report?

Yours sincerely,

D.W. Barron.

Mr. Andy Mickel, Pascal Users Group.
December 10, 1975

Christian Jacobi  
Institut fuer Informatik  
P.T.H. - Zentrum  
CH-8092 ZURICH  
Switzerland

Dear Chris,

How are you?

We are curious to know the results of the Pascal-P questionnaire so that they may be printed in the Pascal Newsletter. Could you please help us all out?

Also, how many sites are on your distribution list? We have a rough idea from George Richmond's implementors list in Newsletter #4, but I'm sure that PUG members would find this information interesting. Also how many Pascal-6000 sites do you distribute to?

Be sure to keep our contact up to date regarding prices and the distribution policy printed in the Newsletter for both Pascal 6000 and Pascal-P.

Thanks a lot!

Sincerely,

Andy Mickel, Editor
Dear Andy,

After the great "standards outcry" of 6, I want to contribute my two cents worth.

First, I fully believe that Standard Pascal, as defined by Prof. Wirth in the latest Report, is defendable in terms of the two objectives stated there: a good teaching tool, and reliable and efficient implementation. It is not surprising then that Standard Pascal should be frozen by its author. I think we must accept the fact that any politically accepted changes or extensions leave us with a different language (perhaps a "Standard Production Pascal").

Since Pascal is being used for purposes not encompassed by Prof. Wirth's objectives (e.g., applications, production, and even commercial processing), there is need to reassess the language. It is important however to distinguish clearly between any new version of Pascal and the Standard Pascal in the Report. To do any less is inconsiderate of a creative person whose work will remain relevant to further development.

Second, as an applications-oriented user, I do see the need for extensions (and possibly changes). The resulting language must have a "political standard" (e.g., ANSI), and thus a standards committee is inevitable. A relatively centralized body is necessary for effective discussion and debate on detailed issues, as well as for providing a formal mechanism for language maintenance as Mike Schneider discussed in 6. What I fear is that in the formation of a committee is the power of deciding what the "standard" will be. I quote the word standard because of the difference between official policy and what we fondly refer to as the real world. We want the standard to be "good", and we want the standard to be enforced. I don't believe that the formation of a committee of benevolent despots will assure either outcome.

A "bad" standard should not be enforced, and hopefully with enough intelligent and independent-minded users and implementors a "bad" standard cannot be enforced. On the other hand, users will create their own non-standard if the official position does not account for real or perceived need for change. There is ample evidence to show that Pascal is both portable and mutable, suggesting that supply will grow to meet demand whether officially standard or not.

Regardless of how a standards committee may be established, I think that it should accept suggestions for changes and extensions from users, and if necessary should shape the suggestions into fully developed proposals for consideration and decision by the entire user community. Under no circumstances should a committee have the authority to decide what will and will not be officially standard. Its purpose should be to organize a facility to handle the mechanics of standardization, and to filter or reshape fragmental or poorly thought-out suggestions (i.e., quality control).

Each suggestion considered by the committee must be weighed against many criteria, and these deliberations should be included in the resulting proposal. Each proposal should also include complete definitions, examples, implementation recommendations, proof techniques, and perceived impact on the entire language.

One of the few points on which I disagree with Richard Cichelli (6) deals with committee representation of implementors. As I see it, one of the prime motivations for a committee's existence is to assure that officially standard features are feasible -- i.e., implementable on (nearly?) all machines. My experience suggests that users do not appreciate in all cases the implementation difficulties inherent in their suggestions. What better way is there to represent implementors than through such a committee? The committee should combine the skills of the language designer and the language feature designer (C.A.R. Hoare, "Hints on Programming Language Design") with those of the implementor.

The representation of users is of course very important. The Pascal Newsletter provides an excellent forum both for the dissemination of proposals, and for the reactions and views of users. I doubt that users can be better represented than by having the fate of each proposal be determined by a general vote.

Finally, I'm afraid that we as users are most vulnerable right now because of the lack of any formal mechanism for controlling change in the language. As serious users of Pascal we have an investment in code. I hope that all currently active implementors are considering very carefully the costs of deviating from the current standard (User Manual and Report). Incompatible implementations can only have the effect of devaluing our investments.

Optimistically,

James F. Miner
We must again thank Tim Bonham and all the many who have responded to his letters. Things seem to be on track. This is what we have to print. If you don't find what you are looking for, check newsletters #5 and #6. 

**Note:** New implementors: We don't know who you are, so please keep us informed; use the checklist below.

**CHECKLIST**

- Names, addresses, and phone numbers of implementors and distributors.
- Machine(s) (manufacturer, model/series).
- Operating system(s), minimal hardware configuration, etc.
- Method of distribution (cost, magnetic tape formats, etc.).
- Maintenance policy (for how long? future development plans? accept bug reports?).
- Fully implements Standard Pascal? (why not? what's different?).
- Compiler or interpreter? (written in what language? length in source lines, compiler or interpreter size in words or bytes, compilation speed in characters per second, compilation/execution speed compared to other language processors (e.g. FORTRAN)).
- Reliability of compiler or interpreter (poor, moderate, good, excellent?).
- Method of development (from Pascal-P, hand coded from scratch, bootstrapped, cross-compiled, etc.; effort to implement in person months, experience of implementors).

---

**PASCAL-P**

Pascal-P is a "portable" compiler for Pascal generating code for a hypothetical "stack machine". It comes on tape as a kit and may be used to bootstrap compilers onto real computer systems. Not much has changed with Pascal-P distribution (we have yet to receive a copy of George Richmond's new order form). Please refer to PUGNS #5 and #6 especially #6 for ordering information.

---

**IMPLEMENTATION NOTES**

(SOURCE INFORMATION, PROPOSALS FOR EXTENSIONS TO STANDARD PASCAL, BUG REPORTS, PROGRAM WRITING TOOLS, ETC.)
IMPLEMENTATION NOTES
(SOURCE INFORMATION, PROPOSALS FOR EXTENSIONS TO STANDARD PASCAL,
BUG REPORTS, PROGRAM WRITING TOOLS, ETC.)

DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITY OF COLORADO
Boulder, Colorado 80309 USA
1 September 1976

PASCALJ

Since our February distribution of PASCALJ we have concentrated upon
cleaning up the compiler itself and resolving some small anomalies in
the definition of Janus. This work is reflected in the current
distribution.

We have received feedback from several groups who have attempted
implementation of PASCALJ via a full bootstrap. This feedback indicates
that such an implementation is very time-consuming with current tools,
and may well lie outside the reach of most people. Our own experience
indicates that this view is substantially correct: PASCALJ is not
really portable at the present time.

During the next months we plan to concentrate upon improvement of the
bootstrap. Our basic approach is to provide tested tools which will map
Janus code into that of a simple machine which is easily realizable but
not perfectly efficient. We hope that by February we will have brought
the bootstrap within reach of most potential implementors.

One recurring theme in the letters we have received is that the
syntax of Janus is too strongly biased towards STAGE2 processing. This
is a valid criticism. We have derived an LL(1) grammar for Janus, in
which the basic symbols can be extracted by a simple lexical analyzer.
Examples of changes are:

DISP3 becomes DISP(3)
AINT 5 becomes A INT 5

The September compiler does not incorporate these changes; we hope to
release them in February also.

- Software Engineering Group

A. IDENTIFICATION

Program name: PASCALJ
Authors: B.W. Pavenel, C.B. Mason
Date: 1 September 1976

B. GENERAL DESCRIPTION

PASCALJ is a compiler which translates the high-level language
PASCAL to the intermediate language Janus. This compiler was
originally written in PASCAL, and used to translate itself to
Janus. The original PASCAL program is included in the Janus text
as comments, and may be extracted by selecting only those lines
which have a period in the first character position. All other
lines should be ignored during extraction, and the periods in
position 1 should be replaced by spaces.

PASCALJ is defined in the following book, which is available from
its publisher (Springer-Verlag):
Jensen, K., Wirth, N., PASCAL User Manual and Report

It is necessary to indicate the character set of the host
computer on which PASCALJ is to be implemented, as the compiler
must provide a mapping from this character set to its own internal
representations. We are providing several standard character sets
from which one should be chosen; should some other character set be
required please send a detailed collating sequence for it.

Character sets available:

- ASCII (full set, 96- or 64-character subset)
- EBCDIC
- CDC display code

C. DOCUMENTATION

<table>
<thead>
<tr>
<th>Document</th>
<th>Quantity</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEG-76-1 A Preliminary Definition of Janus</td>
<td>5 oz</td>
<td>$4.00</td>
</tr>
<tr>
<td>SEG-76-2 PASCALJ Implementation Notes</td>
<td>2 oz</td>
<td>2.25</td>
</tr>
<tr>
<td>SEG-76-1 Janus Memory Mapping: The J1 Abstraction</td>
<td>2 oz</td>
<td>2.25</td>
</tr>
</tbody>
</table>

D. TEXT (Includes STAGE2)

- 7-track magnetic tape (1200-foot reel) 1.75 lb 28.00
- 9-track magnetic tape (1200-foot reel) 1.75 lb 39.00

(Deduct $7.00 from tape cost if you supply a 1200-foot reel. We
will accept longer reels, but we must charge more for postage.
Overseas orders add cost of postage and specify type of shipping.)
CONCURRENT PASCAL (no new information - see Newsletter #6)

Modula (maybe more information in the next newsletter)

SOFTWARE WRITING TOOLS

Mike Ball (distributor and maintainer of a Univac 1100 Pascal compiler) phoned recently to advocate the formation of a clearing house for Pascal Software Writing Tools similar to what we've requested in the Editor’s Contributions in #6 and this issue. He confesses that his 3 compilers keep him too busy to perform this service and that it will have to be someone else. He would like to contribute some programs to the pool (and I'm sure that other people would want to do likewise). One of these programs is a "source code configurator". Mike cited the problems of character sets and a coordinated standard for external and internal documentation. (* Perhaps we also need a simple editor to facilitate updates for these programs. *)

BURROUGHS B-6700/B-7700

Last newsletter a vague reference was made to a technical report about A. H. J. Sale's implementation in Hobart, Tasmania, Australia. The full reference is: "Burroughs PASCAL - Some Implementor's Thoughts", Arthur Sale, Department of Information Science Report 76-2, University of Tasmania, 14 pages.

In its contents are sections on: Principles; Design Decisions; Lexical Tokens; Syntactic Features; Machine Dependency; Run-Time Actions; Compiler Options; Input and Output; Regrets; Closing Remarks; and References.

CII Iris 80, 10070

In a letter dated November 11 to Tim Bonham, Didier Thibault, 8 rue d'Auteuil, 75016 Paris, France, wrote: "...joined is the description and the details about the status of our project: Pascal compiler for CII Iris 80, 10070 computers. All this information may be of interest to PUG members. You may publish it in the next Pascal newsletter. I am sending a copy of your letter to the maintenance group in Toulouse so that they may get in touch with you...."
IMPLEMENTATION

This is a full compiler generating object code for the
linkage editor. The compiler consists of

a MONITOR: written in assembly language (size: 2 words of 32 bits)
It links the PASCAL program to the operating system and
controls the execution of the PASCAL program. All operating
system dependencies are located in this monitor. To get
the compiler available on other operating systems, the
rewriting of this monitor is necessary.

a COMPILER program: written in PASCAL language itself, it consists of
4800 lines of PASCAL program. It is a one pass compiler
with top-down syntax analysis, separate compilation of PASCAL
programs, symbolic post mortem dump output, and many
specific options.

The compiler is fully bootstrapped so that any user
may adapt it easily to its specific need (table sizes,
specific features)

a PASCAL LIBRARY used by the linkage editor

EXPECTED REQUIREMENTS to run the PASCAL system:
minimum 30 K word in the overlay version of the system
45 K word without overlay

COMPILATION SPEED (measurement on CTI IRIS 80 non-processor)
1000 PASCAL lines/sec
2400 char/sec.
(POTRAN speed: 1000 char/sec.)

EXECUTION SPEED: dependent on the program profile

<table>
<thead>
<tr>
<th>POTRAN</th>
<th>PASCAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix multiplication</td>
<td>1</td>
</tr>
<tr>
<td>recursive program</td>
<td>1</td>
</tr>
<tr>
<td>character count</td>
<td>1</td>
</tr>
</tbody>
</table>

RELIABILITY OF THE COMPILER
The actual release is release 3. This compiler
has been tested over 2 years in 25 different computing centers.
The reliability is good-excellent.
Dear Pr. BONHAM,

In addition to the letter by D. THIBAULT dealing with the PASCAL/IRIS-80 compiler, I give you the following information:

I am a member of the IRIA's project SFER which has, among other works, maintained, developed and distributed the PASCAL compiler since July 76 and will go on until January 78; it is used by about 30 centers mainly in the fields of software teaching and development.

The compiler, including all source programs -PASCAL and assembler code- is freely available; you must just send a tape (mini, if possible) to the above address.

Transpositions to other systems can be an easy task; it depends on the binary code compatibility. The CII-IRIS 80 machine uses the same machine language as the XDS-Sigma 7. A successful transposition has been made by Mr. TAKEI, Tokyo University, to an XDS-Sigma 7, running under the BPM monitor.

Usage reports are encouraged and reflected into the following releases of the system.

The compiler accepts standard PASCAL (ref. "User Manual and Report" by K. JENSEN and N. WINTH, Springer Verlag 75) with some additional features - separate compiling, compile time initializations, extensions to read and write procedures for use in an interactive environment - and perhaps some minor differences - standard PASCAL is sometimes somewhat evasive and I have no access to a CDC computer.

Another IRIA's project develops a PASCAL subsystem (under Time-sharing system SIRIS 8 TS) designed to provide the users with powerful commands for editing, management (store and retrieve) and exploitation of PASCAL programs. The first release of the system may take place at the beginning of 1977.

Please, contact me for further information,

Your truly,

P. MAURICE

P.S. I sent my contribution to PUG a couple of months ago and did not receive any newsletter issue.
Prerelease Implementation Notice. January 2, 1977

1. Implementors: John T. Easton 612/373-7525
   James F. Miner 612/373-9916
   Jonathon R. Gross

Address correspondence to:
Pascal
SSRFC
25 Blegen Hall
University of Minnesota
Minneapolis, Minnesota 55455


3. Developed under OS/S version 3. Hardware required:
   - KEO-E (EAE with mode B instruction set).
   - KBP-E disk, or other direct access mass storage device with at
     least 131K 12-bit words (e.g., DF32 or RFOS).
   - 32K of core/RAM is highly recommended. Minimum is 16K.

4. Distribution, documentation, and maintenance details are not yet available. Hopefully more information will be available in the next Pascal Newsletter.

5. Emphasis has been placed on close adherence to the Pascal User Manual and Report.

6. As with most languages on the PDP-8, Pascal makes use of an interpreter (a modification of P-code) written in PAL8. The compiler and assembler are written in Pascal. All standard procedures are written in PAL8.

7. Execution is roughly comparable to Fortran IV (F4). I/O seems to be faster than Fortran, while computation seems slower. Unfortunately Pascal compilation is much slower than F4. We hope to make some improvements in this area before the first release.

8. Because of the design of the system, the implementation is not suitable for real-time applications. On the other hand, the implementation does provide 131K words of virtual memory for code and store (expandable to 262K).

   Standard Pascal types are implemented as follows:
   
   Standard Pascal types are implemented as follows:
   
<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>1 word</td>
</tr>
<tr>
<td>Char</td>
<td>1 word</td>
</tr>
<tr>
<td>Integer</td>
<td>2 words</td>
</tr>
<tr>
<td>Real</td>
<td>4 words</td>
</tr>
<tr>
<td>Set</td>
<td>8 words</td>
</tr>
</tbody>
</table>

9. The system has been fairly reliable.

10. We have spent about 9 man months so far. It has been useful to cross-compile from the University of Minnesota's CDC Cyber machines.

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

Very truly yours,

S. C. Schwarm, Development Engineer
Engineering Physics Laboratory (Beech Street)

Craig E. Bridge, Engineer
Engineering Physics Laboratory (Beech Street)
E. I. Du Pont de Nemours & Company, Inc.
Wilmington, Delaware 19898

SCS/CEB:pjm
Atch
Our installation has as many problems in getting a working "standard" PASCAL compiler up on our machines as any. All of our efforts so far have been aimed at transporting the work of others to our site. We have no original work available yet.

The efforts involved at this site have been to:

I. Bring up Per Brinch Hansen's SOLO/CONCURRENT and SEQUENTIAL PASCAL compiler on his minimum machine (- nine track magtape). This compiler works reasonably if you are willing to suffer along under the SOLO executive (poor editor/single user). The variable scopes are also limited to GLOBAL and LOCAL, thus voiding some of the capabilities of nested procedures. These compilers are tied extremely closely to SOLO and are nonstandard. We have developed the following to aid others in using SOLO:

A. BOOTSOLO: copies nine track SOLO distribution magtape to SOLO RK03/05 disk cartridge. (Same as distributed SOLO bootstrap but runs on PDP-11/04/10/15/20/34/40/45/50/60/70 with 16K core, console terminal, nine track magtape, RK03/05 disk.)

B. RSOLO: copies files from SOLO RK03/05 disk to DOS V9.20C (possibly V4?) RK03/05 disk. Runs under DOS/BATCH-11 V9.20C (V4?) on 16K PDP-11/04/10/15/20/34/40/45/50/55/60/70 with 16K core, two R103/05 disk drives, console terminal.

C. READOS: copies files from DOS/BATCH-11 V9.20C (possibly DOS-11 V4?) disk to SOLO disk. Runs under SOLO and PDP-11/45/50/55/60/70 with 48K core, floating point unit, two RK03/05 disk drives, console terminal.

II. Bring up University of Illinois PASCAL compiler under MINITAUR operating system (bastard son of DOS-11 V4?). This system is nonstandard, it works well, but is ugly inside. Runs on PDP-11/04/10/15/20/34/40/45/50/55/60/70 in 28K core under MINITAUR. DOS/BATCH-11 V9.20C can read/write files onto MINITAUR file structured device (not the other way however). System comes up in one evening.

Documentation is scant, however, and the command syntax for batch streams is entirely different from DOS/BATCH-11 which came in DOS-11 V8.0 and after.

III. Bring up DECUS PASCAL-11 from Lucien Feiereisen, et al (PASCAL to JANUS through Stage 2 macros to MACRO-11).

11/10/76
Came up in one afternoon. This implementation has a rather weak stomach. It will not digest anything but small syntactically correct (as per this implementation as opposed to "standard") programs. Most errors end up as F342 (addressing exceptions) without hope of further diagnosis. This particular implementation runs on a PDP-11/34/40/45/55/60/70 under DOS V9.20C with 64K core.

This implementation has given DEC a sour taste on PASCAL (much to our dismay).

IV. Bring up a PCODE-2 to MACRO-11 translator in hopes of bootstrapping the PCODE PASCAL compiler. This approach got bogged down in the problem of knowing what size the stack elements that were pushed, popped and operated on. The word and byte alignment problem was the straw that broke PCODE-2's back. We have just recently received the PCODE-3 and 4 release which promises to solve these problems. Work will continue in this direction.

V. Bringing up several of the PDP-10 PASCAL compilers (all of which work well). Don't wish to elaborate here since newsletter 5 covers this well.

S. C. Schwarm/C. E. Bridge: pjm
11/10/76
The following contains a summary of the PASCAL-cross-compiler for the PDP 11 as of November 1976. The reader is expected to be familiar with the publication in "Software, Practical Experience, Vol. 6, 109-116 (1976).

With regard to the definition of the language PASCAL the following restrictions hold:

Files are not implemented, apart from the standard files input and output which are always linked to the keyboard/printer-device.

Packed data structures are only implemented for:

- one-dimensional character arrays (packing always takes place), and
- one-dimensional boolean arrays (packing optional, one boolean per bit).

There is no procedure DISPOSE to delete individual dynamically created structures, but there is a MARK/RELEASE pair which suffices when allocation and deallocation takes place in nested fashion. Only local jumps are allowed.

The following extensions have been implemented:

- function results can be of non-scalar type,
- arrays with unspecified bounds (but specified index-structure) can be passed as arguments to procedures,
- several standard procedures have been added, notably two procedures to obtain and set the value of specified memory-locations. These procedures are intended for communication with the device registers. Undiscriminated use of these procedures may endanger the implementation,
- procedures may be declared in the outer program block to be associated with specified interrupt-sources. Apart from the level of declaration there are no restrictions on interrupt procedures. Of course interrupt procedures cannot be called in the conventional sense,
- a string parameter type has been introduced in which one dimensional character arrays or substrings thereof may be passed as actual parameters. Such strings and their constituent characters are considered as "read only". The most important application is in passing strings to procedures which may in turn pass on these strings to "write", which up till now was the only procedure to accept strings of unspecified length.

Current Experience.

Experience gained so far shows the implementation quite fit for real time applications.

Recently a number of PASCAL routines have been developed and implemented for the implementation of parallel processes. We mention: queue-operations, process switching, starting of independent processes, P- and V-operations, monitors (à la Hoare) and conditional critical regions, device-drivers, interrupt handlers, asynchronous I/O. These PASCAL-routines have been based on three routines that were written in PDP 11 assembler and consisted of appr. 50 words. These routines performed:

- register switching, interrupt enabling and disabling, and a bit of address manipulation to make the address of a procedure available to the PASCAL environment. These three routines will soon be implemented in the form of standard-procedures.

Future Developments.

A full implementation of the file-concept to run under RT 11 is expected to be completed shortly. This implementation will only need minor revisions in order to become operational under other DEC operating systems.

In a next phase the development of a simple multiuser operating system centred around a file system (fully implemented in PASCAL) is foreseen.

C. Bron.
November 12th. 1976.
Twente U. of Technology,
P.O.Box 217,
Enschede.
The Netherlands.
Dear Andy:

We recently received your letter to David Bennett here at PAR, asking about the status of our PASCAL compiler implementation efforts. Here it is:

1) We attempted to construct one from the ill-fated PASCAL P compiler kit, but:
   a) The interpreter subroutines plus threaded code would not fit in 32000 words, our maximum allowed program size, and system overlay facilities were not applicable to interpreter code which looks like data.
   b) The P assembly language was not very suitable. It required tag fields with data and had several other major faults.
   c) My bosses weren't ready to spring for the newer version of the kit after being burned with the first kit.

2) I analyzed what was lacking in the kit and decided it was not nearly as portable as it could be if it translated PASCAL into some language for which everybody already has a translation to machine language (you guessed it -- FORTRAN).

3) I spent 3 months thinking up methods by which PASCAL might be translated efficiently into FORTRAN. Then I began writing this compiler in PASCAL, translating it by hand into its equivalent FORTRAN and debugging the FORTRAN. I am about halfway (with about 3-5 months of work remaining) towards having a full PASCAL compiler (including recursive procedures) which will produce reasonably efficient FORTRAN.

This is a solo effort on my part, with two goals:

1) Obtain the first PASCAL compiler (to my knowledge) which runs under RSX-11D on the PDP 11/45.

2) Produce a truly portable PASCAL compiler (i.e., one that can be set up with no more work than transporting the compiler source in PASCAL and the compiler object code in FORTRAN to another computer.

Here's a list of information about the project in the same order you requested it.

1) Implementor - Michael N. Condict
   PAR Corporation
   On The Mall
   Rome, N.Y. 13440
   Phone - (315) 336-8400

2) Machine - DEC, PDP 11/45

3) Operating Sys. - RSX-11D
   Minimal Config. - Same as for RSX

4) Method of Distribution - None for at least 5 months.

5) Documentation - Not yet.

6) Maintenance - Not yet.


8) Type of Program - PASCAL to FORTRAN translator, expected to be about 5000 FORTRAN source lines excluding comments, and about 3000 PASCAL source lines. Will be one pass and thus run rings around FORTRAN compiler.
Reliability - Will not be distributed until it is.

Method of Development - Initial version of each procedure was written in PASCAL then hand translated into a "pretty" FORTRAN preserving all identifiers by use of the PARAMETER statement (allows symbolically named constants). The FORTRAN program (currently 2500 source lines and half finished) was debugged after each addition to it (I am using both top down development and top down debugging by writing "stub" procedures, later replaced with real ones). When the compiler is finished or can compile itself, it will be restored to its original PASCAL in a massive, manual, inverse translation, and run through itself, thus completing the bootstrap.

Man Months - I expect this project to require between 6 and 9 man-months (with 1 man devoting half his time). It has currently consumed about 4 man months.

Experience of Implementer - I built a compiler for a subset of PASCAL as a class project once, but this is already the largest program I have ever attempted.

I hope this answers all questions you may have about our implementation efforts. We are still interested in learning of any existing full PASCAL compiler which runs under RSX on the PDP 11/45, if you are aware of one.

Sincerely,

PAR Corporation

Michael N. Condict

PASCAL User's Group

Andy Mickel

14 December 1976
The interpreter Bob and Jim developed does have a few interesting aspects, though. Approximately 14K of memory is used to execute the interpreter, including 14K words of 'stack,' 12K words of P-CODE, and 19K words of code for the interpreter itself. Of course there is a lot of overlaying involved in placing all of this into 14K, and our implementation is, therefore, fairly slow. But since our primary use of PASCAL is as a research tool we can live with this.

We do have a few 'enhancements' built into our interpreter. For example, we have a trace facility that can be controlled through a code in a PASCAL program. This was intended primarily for debugging the interpreter, but has also proven useful in debugging programs written in PASCAL. The output of the trace is variable, based on the type of instruction being interpreted. Basically the output consists of the operation code, operand fields, stack pointer, program counter, and top four locations of the stack. As with most trace facilities, output can be turned on and off at the whim of the programmer.

Another optional output of our interpreter is what we call a 'profile.' This consists of two lists. The first indicates the number of P-CODE instructions, by type, and the percentage of code that occursances of each instruction type constitute. The second list shows, by instruction type, the number of times an instruction of that type was 'executed,' and the percentage of 'execution time' spent in instructions of that type.

It was intended that this profile serve two purposes. First of all, if we ever got around to optimizing our interpreter we would have some idea of most frequently used instructions, etc. Secondly, we could project actual execution time of PASCAL programs on either the FOX 1 or some other system by building tables of instruction execution time. At this time we have done nothing more than produce profiles, however.

We have, at present, four PASCAL programmers (or should I say programmers who can write PASCAL programs) at our installation. I wish I could tell you we use PASCAL heavily, but that's not the case. The higher level language supported by our product systems happens to be FORTRAN and so we are somewhat locked into it. Other than the occasional program one of us might write we are, in fact, doing very little with PASCAL.

I hope the above helps to clarify our present position. Should any change in the status of PASCAL at Foxboro occur I'll let you know. Until then, we hope to keep abreast of what's happening with PASCAL through your newsletter.

Sincerely,

Thomas G. McGinty
Corporate Research

---

IBM System 360/370

STANFORD UNIVERSITY

STANFORD LINEAR ACCELERATOR CENTER

December 20, 1976

Timothy Bonham
PASCAL Implementations
University Computer Center
227 Experimental Engineering Building
University of Minnesota
Minneapolis, Minn. 55455

Dear Tim:

Your letter of inquiry about the SLAC PASCAL Compiler was (eventually) directed to me and I apologize for the (very) long delay in the process of responding.

Our PASCAL Compiler for the IBM 360/370 system is based on the P-compiler (the P2, May 74 version) and, with a few minor extensions, implements exactly the same language.

During the bootstrap process, the output of the P-compiler was translated by a set of IBM Assembler-H Macros into 360/370 Assembly language and then into object code. The resulting code was overinflated and fairly inefficient. Currently, we have a post processor (a so-called P-Translator) which is a 2500 line PASCAL program that translates the output of the P-compiler into either 360/370 assembly language or directly produces a standard OS object module.

In addition to fixing some of the obvious bugs (and some not so obvious) in the P2 Compiler, we had to modify the compiler and the syntax of the P-Instructions in order to specify the type of the operand(s) of some instructions (in particular Load, Store type operations) before a reasonable translator to 370 code could be written. In other respects, the Syntax Analyzer is almost as it was in the P2 compiler.

The P-Translator is a one-pass, no lookahead translator which could be incorporated into the compiler, or ideally, communicate with it in coroutine fashion but, for simplicity, we run it as a post processor to the P-compiler with somewhat increased I/O overhead. The current version of the P-Translator (like the P-Compiler) does not provide run time checking of indices or subranges.

Sincerely,

Thomas G. McGinty
Computer Section
Corporate Research
Some statistics:

<table>
<thead>
<tr>
<th></th>
<th>P-Compiler</th>
<th>Post Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>lines of source code</td>
<td>4000</td>
<td>2500</td>
</tr>
<tr>
<td># of P-Instructions</td>
<td>15000</td>
<td>52000</td>
</tr>
<tr>
<td>bytes of 370 instructions</td>
<td>76000*</td>
<td>52000*</td>
</tr>
<tr>
<td>time to compile the compiler</td>
<td>10 sec.+</td>
<td>5 sec. + (total 15 sec)</td>
</tr>
<tr>
<td>processing rate</td>
<td>400 lines/sec.</td>
<td>800 (source) lines/sec.</td>
</tr>
</tbody>
</table>

*this includes the I/O interface and conversion routines
+timed on the 370/168 with 16k cache memory.

The Compiler/Post-processor is capable of compiling (bootstrapping) itself in a 130k byte region, 24k of which is returned to the operating system for I/O buffers.

The entire system is available to the public (as is) and we are working on further optimization of the object code.

Once again, sorry for the delay in our response. Please feel free to write me if you need more detailed information.

Sincerely,

Sassan Hazeghi
Computation Research Group
Mike Ball at the Naval Undersea Center, San Diego reports that Paul Fisher at Kansas State University has a slow version of Brinch Hansen Pascal on the Interdata 8/32 which ignores lower case characters. Mike is currently implementing Pascal himself on the 8/32, changing Brinch Hansen's compiler to a 9-pass highly optimizing compiler.

Mike passed on the observation that it took Brinch Hansen's compiler three times as long to bootstrap but had far fewer errors than the P-compiler. He would like to know other people's implementation experiences. (*See the CHECKLIST, item 10.*)

**UNIVAC 90/70** (*No known implementations; see Here and There under Hopkins.*)

**UNIVAC 1100 SERIES**

Mike Ball of the Naval Undersea Center, San Diego, reports that he has received many more requests than he expected for his Univac 1100 Pascal compiler. He has talked to Fischer and LeBlanc at the University of Wisconsin, Madison, WI about their compiler. They are using his to help them write a diagnostic, checkout compiler based on LR(k) scanner with variant record and pointer checking. This should be useful for students. They are also concentrating on code optimization.

We received the following more complete news of the 1100 implementation from DIKU, University of Copenhagen. It appears that for this compiler, the source code is not released. However, the documentation appears good; a nice (unfortunately faint) 19 page machine retrievable manual was sent to us entitled: "A Pascal Compiler for Univac 1100 machines" by J. Steensgaard-Madsen and Henrik Snog at the Datalogisk Institut, University of Copenhagen. It is in the form of a supplement to Pascal User Manual and Report. This compiler has 6-bit characters as standard type CHAR but does supply an additional type ASCII. ALFA is then a predefined type of packed array[1..12] of CHAR and there is another predefined type HALFA which is packed array[1..6] of CHAR (or 6-6 bit characters per 36 bit word). Set sizes are only 72 elements as compared to 144 in Mike Ball's compiler.
Pascal 1100A:
An extremely fast compile-and-go batch system.

Pascal 1100S:
A compiler generating comprehensible assembler code. A true extension of version A.

Common characteristics:

- Deviations from Standard Pascal:
  A. Any TEXT variable F will after RESET(F) fulfill:
     EU(F) = FALSE and EOLN(F) = TRUE
     (This allows a program to open a dialog with a user).
  B. Parameters of formal procedures and functions may be
     of any kind, but specification is required.

- Restrictions:
  A. File components containing files should not be used.
  B. Standard procedures cannot be passed as actual parameters.
  C. The value of ORD(X) for X belonging to the base type of
     a set must be in the interval 0..71.
  D. The first operation (in the dynamic sense) on any file
     must be REWRITE.

- Extensions:
  A. A scheme for exhaustive specification of parameters.
  B. Constant definitions according to the syntax
     <identifier> : <type> = <values>;
  C. An OTHERWISE clause in a CASE statement.
  D. A LOOP ... EXIT ... END statement.

Machine dependent facilities:
- Handling of both Fieldata and ASCII characters and files.
- REAL is double precision always.
- Dynamic association of external files.
- Smooth cooperation with the operating system.

Pascal 1100S features:
- External procedures and functions may be written in Pascal
  or (ASCII) Fortran.
- Inclusion of assembler code is possible.

Compiler performance:
- Both compilers require a bit more than 40K words to execute.
- Compilation speed is roughly 100 lines per SUP second (SUP =
  Standard Unit of Processing, defined by UNIVAC).
- Compiler reliability is excellent.
- Compiled programs run efficiently compared to other processors.
Xerox Sigma 6/Sigma 9

PASCAL - SIGMA (2.0)

NOTES ON THE DISTRIBUTION PACKAGE

Distribution includes a magnetic tape containing the latest system release and a package of documentation. All programs are commented in English but documentation is in French (except for the “Program description”).

Distribution costs, which are $250.00, do not imply any responsibility or maintenance service on the part of the distributor, implementor or the Université de Montréal. Distribution costs may be paid by sending a check (payable to Pierre Desjardins) to the address below.

PIERRE DESJARDINS
Université de Montréal
Informatique
C.P. 6128, MONTREAL 101
Québec, CANADA

Release: September 1976
Implementor: Pierre Desjardins
Distributor: Pierre Desjardins

19 November 1975

Jørgen Steensgaard-Hansen
Datalogisk Institut
Københavns Universitet
Sjællandsvej 41
DK-2200 København
DENMARK

Dear Jørgen,

Thank you for the information on your Univac 1100 Pascal. We received it November 15, too late to be printed in Pascal Newsletter #5 (which is 93 pages). It will definitely appear in #7.

Your letter indicates that there was some confusion on our part that I would like to clear up. First, we received the subscription from your library in mid-October. The mailing of issue #5 has been delayed because we have been negotiating with Judy Mullins at the University of Southampton, England about European mailing. This issue has now been air mailed directly to your library, but the library will receive further issues from Southampton.

Second, we sent letters similar to the one you received to over 30 other known implementors. We wrote two versions of the letter, one for PUG members, and one for non-members. We have no record of your membership, so we apparently sent you the wrong version. Please accept my apologies for this mistake.

I would like to take this opportunity to personally invite you to join the Pascal User's Group. This is an informal group of users, teachers, implementors and just plain fans of Pascal. We now have 528 members in 22 countries. We have published and mailed issue #5 (64 pages). Issue #6 is now at the printer. I am enclosing a membership coupon.

Thank you again for the Pascal 1130 information. It is important to us that we keep the Implementation Notes section current and correct. Your documentation will certainly help in Newsletter #7.

Sincerely,

John P. Strait

John P. Strait

Enclosures.
COMPILER CHARACTERISTICS

1. The compiler is a 6220 lines Pascal-Sigma program (Pascal-Sigma corresponds to Wirth and Jensen's "Pascal: user manual" together with the restrictions cited below);

2. it was obtained by cross-compilation on a CDC CYBER74, using the Nov.1972 version of Pascal from Zurich;

3. without any prior knowledge of Sigma machines, the entire programming system was completed in 18 man months;

4. reliability is considered "good to excellent";

5. language restrictions are as follows:
   - sets are limited to 32 elements,
   - standard procedures (functions) are not allowed as actual parameters,
   - array, record and file type cannot have file type components,
   - string constants cannot be defined in the const part of the declarations.

6. non-standard extensions to the language include:
   - keyed files,
   - ghost variables

7. peak code size is 25K;

8. self-compilation requires 35K (peak) of core and executes at the rate of
   - 600 lines/min. on Sigma 6 (BPM/BTM)
   - 1200 lines/min. on Sigma 9 (CP-V)

9. generated code takes the form of a file of relocatable object modules (one for every procedure (function)) in "Xerox Standard Object Language".

TAPE STRUCTURE AND FORMAT

Tape format is: 9 tracks, 800 bpi, binary.
The tape is structured much like a standard Xerox processor distribution tape (labelled tape in account :SYSGEN). As specified in the "Program Description", file $S$CONTENT displays the contents of the tape.

MAINTENANCE POLICY

As specified earlier, distribution costs do not include "on-demand" maintenance service. However, bug reports are welcomed so that once they have been seen to and a few of them have been accumulated, update sheets could be sent to Pascal-Sigma users.

Bug reports should be as explicit as possible as to the nature of the bug and the environment in which it occurred.
Except for the "Program Description" document, all other documents are written in French.

Here is a list of all documents provided in the distribution package together with a brief description of their use:

1. "Program Description", contains information related to the installation and maintenance of Pascal-Sigma.

2. "Manuel d'utilisation....", is the user's manual. It gives information related to the particularities of Pascal-Sigma: restrictions, extensions and use under BPM/BTM.

3. "METAPASC: ......", will be of interest to people expecting to use external procedures or functions, written in Meta-symbol, inside a Pascal program. METAPASC provides a set of macro-procedures destined to relieve the user of interface details: declaration of parameters and local variables, parameter passing mechanisms (generation of data-segments), retrieving from (storing in) the data-segment by symbolic reference, ....

4. "Pascal 2 - Sigma: un système de programmation Pascal", provides a description of the 'functional structure' of the Pascal-Sigma compiler: the major logical components of the compiler are explained together with the way by which they communicate information. This document was written with the following intention: to provide a description of the major parts of a large program (the compiler) and a global view of its strategy, without annoying the reader with too many low-level details (which he can easily investigate in the source text once he understands the structure).
PASCAL USER'S GROUP

USER'S

GROUP

Clip, photocopy, or reproduce, etc. and mail to: Pascal User's Group
c/o Andy Mickel
University Computer Center
227 Exp Engr
University of Minnesota
Minneapolis, MN 55455
(phone: (612) 376-7290)

Please renew my membership in the PASCAL USER'S GROUP for the next Academic Year ending June 30. I shall receive all 4 issues of Pascal Newsletter for the year. Enclosed please find $4.00. (*When joining from overseas, check the Newsletter POLICY section for a PUG "regional representative".*)

Please send a copy of Pascal Newsletter Number ___. Enclosed please find $1.00 for each. (*See the Newsletter POLICY section for issues out of print.*).

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

You messed up my address. See below.

Enclosed are some bugs I would like to report to the maintainer of the ________________ version of Pascal. Please forward it to the appropriate person so that something can be done about it.

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 Newsletter.

None of the above. ______________________________

______________________________

______________________________

______________________________

Other comments: From: name ________________________________

address ________________________________

phone ________________________________

date ________________________________

(*Your phone number helps facilitate communication with other PUG members.*)
return to:

University Computer Center
University of Minnesota
227 Experimental Engineering Building
Minneapolis, Minnesota 55455 USA

return postage guaranteed