FROM THE EDITOR

The fourth newsletter is long overdue, the third being published in February 1975. There have been many significant events that need announcing. The highlights are:

- Release 2 of PASCAL 6000-3.4 has been made available by Dr. Urs Ammann at Eidgenossische Technische Hochschule in Zurich, Switzerland. It is also available from Mr. George Richmond at the University of Colorado Computing Center in Boulder, Colorado and Mr. Carroll Morgan at the Basser Department of Computer Science, University of Sydney, Australia. Many improvements have been made over Release 1 and future developments are promised. See page 73.

- Mr. Carroll Morgan of the Basser Department of Computer Science at the University of Sydney, Australia has kindly agreed to distribute ETH Pascal and portable Pascal for Australia and neighboring regions. Interested parties should contact Mr. Morgan for more information.

- Mr. Andy Mickel of the University of Minnesota kindly agreed to take over editorial control and publication of the Pascal Newsletter commencing with issue Number 5 in September 1976. He is also organizing a User's Group. See pages 88 and 89.

- An improved portable Pascal has been released from ETH, Zurich. See page 81.

- News of Pascal compilers for numerous machines has been received. See pages 96 and following.

- An expanded bibliography of Pascal literature has been compiled. See pages 100 and following.

I have enjoyed producing the first four newsletters but I have found it to be very time consuming. I am grateful to Andy for taking over these duties with enthusiasm. I will continue to distribute Pascal in North America thru the University of Colorado.

~ George Richmond

Technische Hogeschool Twente
Enschede, Nederland
postbus 217

Comment on an extension of PASCAL

In a recent PASCAL-newsletter (#3) we find a proposal by N. Wirth regarding "A generalization of the READ and WRITE procedures". The proposal, which has been implemented in the PASCAL 6000-3.4 compiler, considers the procedure

read(f,x) to be synonymous to the sequence:
x := f↑ ; get(f) , and
write (f,x) to be synonymous to the sequence
f↑ := x ; put(f)

It is my opinion that standard procedures should only be introduced when

a) their body cannot be described in the language PASCAL, or when
b) there exists an essentially faster mapping of the body directly onto the machine-code (e.g. one instruction that does the job), than the compiled code of the PASCAL-body admits.

Standard procedures of type a) should be a standard for the languages as such.
Standard procedures of type b) should be standard for a particular configuration only, in order to give the programmer a convenient form of access to particular aspects of the machine at hand.
When transferring a program based on type b) standard procedures to another machine, the portability is ensured by supplying the appropriate procedure-declarations.
Procedures that do not satisfy any of the above criteria may nevertheless be placed for convenience in a library of (basically) source-code procedures.
In line with the above philosophy we have decided not to implement the arithmetic functions like sin, atan, ln etc., in our PASCAL-version for the PDP/11 series. Apart from the relief for the builder of the system who has to implement these routines in machine code, one may well suppose that an abundance of standard procedures can be disadvantageous to the compactness of both compiler and runtime system.

Now let us take a look at the file-proposal with the above in mind. Then the proposal only satisfies criterion (a). This criterion is satisfied

since if the procedures were to be described in PASCAL the type of the parameters f and x had to be fixed.
It would, however, be a very minor job to declare the procedures "read" and "write" to be specific for a particular type of parameter. The bodies would all be equal, only the parameter specification having to be adapted. This seems hardly a problem since the number of different file types that are dealt with in one program will be quite limited.
My suggestion is that - if the proposed standard procedures are not available - one simply declares:

```
readplop (f : file of plop ; x : plop);
begin x := f↑ ; get(f) end
```

and

```
writeplop (f : file of plop ; x : plop);
begin f↑ := x ; put(f) end
```

There is, however, a much more important aspect of the current procedures "read" and "write" that asks for a generalization.
Whereas the files they implicitly operate on (input and output) are "file of char" , the actual parameters, may be of arithmetical type and a conversion from or to character sequence is specified by the somewhat extraordinary remaining parameters (in the case of write).
It seems to me that there is a greater need to generalize the procedures "read" and "write" in this respect, viz. making the conversion available for all files of char. One may now visualise how a PASCAL program may build two output streams in parallel, a feature even the compiler could use.
Alternatively, the conversion routines themselves could be made available, but because of rigid data-sizes in PASCAL there is no way of properly dealing with the format specifications. This appears one of the major arguments for considering "read" and "write" as standard procedures.

One might counter my arguments by stating that it is possible to describe "arithmetic quantity ↔ character sequence" conversion wholly in PASCAL but apart from the above arguments concerning the need for flexible data-formats, it is impossible to describe the conversion from a real

quantity to a character sequence in terms of real operations because of the implicit inexactness of real arithmetic. In that case at least a standard procedure has to be supplied for the conversion of a real quantity to (a number of) integer quantities. In other words: if one has a real quantity x, for which . $1 \leq x < 1$, it is not guaranteed that $1 \leq x * 10 < 10$, and therefore trunc (10 * x) may not deliver the correct digit! This example shows, by the way, that "trunc" is a very illdefined function, which should be abolished from programming languages!

C. Bron.
Enschede 25.5.1975.

c.c. Wirth.
Richmond.

LUC FEIEREISEN
INSTITUT F. BIOKYBERNETIK U. BIOMED. TECHNIK
UNIVERSITAET KARLSRUHE
D-7500KARLSRUHE 1
KAISERSTR.12
Gerwany

KARLSRUHE, 30-JUN-75

MR. GEORGE H. RICHMOND
UNIVERSITY OF COLORADO
COMPUTING CENTER
FSRB #3
BOULDER, COLORADO 80302

DEAR MR. RICHMOND,

I THANK FOR YOUR LETTER FROM THE 16-MAY-75. THE PASCAL COMPILER BASED ON JANUS HAS BEEN IMPLEMENTED ON THE PDP-11/45 RUNNING UNDER THE DOS/BATCH OPERATING SYSTEM AND A PRELIMINARY VERSION HAS BEEN RELEASED TO A LIMITED NUMBER OF SITES.

THE AVAILABLE PASCAL-COMPILER (PAS74) IS WRITTEN IN THE LANGUAGE IT TRANSLATES. ITS JOB IS TO ANALYSE A PASCAL PROGRAM FOR SYNTACTIC ERRORS AND TO GENERATE CODE FOR A STANDARD ABSTRACT MACHINE CALLED JANUS [COL73].

THE MACROGENERATOR STAGE2 [WAI73, WEI74] MAPS THIS SYMBOLIC JANUS CODE INTO THE PDP-11 ASSEMBLER CODE, MACRO-11. THE JANUS/MACRO-11 TRANSLATOR IS DEFINED BY SET OF STAGE2-MACROS. AS THE PRODUCED CODE IS ORGANIZED AROUND AN IDEALIZED ABSTRACT MACHINE THERE IS LEFT CONSIDERABLE ROOM FOR OPTIMIZATION. CODE AND DATA ARE SPLIT INTO 2 SEPERATE FILES.

THE FINAL TRANSLATION TO ABSOLUTE CODE IS PROVIDED BY THE NORMAL PDP-11 ASSEMBLER AND LINKER. THE PASCAL LOADER LOADS THE DATAFILE INTO THE USER-DATA-SPACE (32 K MAXIMUM) AND THE CODEFILE INTO THE USER-INSTRUCTION-SPACE (32 K MAXIMUM).

THE PASCAL-COMPILER REQUIRES TO COMPILE ITSELF (PASCAL/JANUS) 64 K WORDS OF MEMORY AND 404 SEC. THE TRANSLATION AND EXECUTION TIME OF PASCAL AND FORTRAN WERE COMPARED: THE WHOLE TRANSLATION PROCESS IS ABOUT THE FACTOR 1.29 THAT OF EQUIVALENT FORTRAN PROGRAMS ON THE PDP-11, WHEREAS THE EXECUTION SPEED IS ABOUT THE FACTOR 0.65.

ALL FEATURES OF THE USED PASCAL LANGUAGE (CLASS AND ALFA VARIABLES, VALUE AND FILE DECLARATIONS, GLOBAL EXITS, ...) ARE IMPLEMENTED EXCEPT FOR PARAMETRIC PROCEDURES. THE FLOATING POINT PROCESSOR IS USED FOR REAL ARITHMETIC AND FOR TEXT-HANDLING (ALFA TYPE). THE I/O CONCEPT INCLUDES CONCURRENCY AND EXPLIZIT OUTPUT CONTROL (GRAPHIC OUTPUT POSSIBLE TOO).

THE PASCAL-11 USER'S GUIDE PROVIDES INFORMATION NECESSARY TO INSTALL THE PASCAL-11 SYSTEM AND TO TRANSLATE AND EXECUTE PASCAL PROGRAMS ON THE PDP-11/45.

THE PASCAL-COMPILER CAN BE MOVED TO ANOTHER COMPUTER WITH A LIMITED AMOUNT OF EFFORT: THE JANUS TRANSLATOR, WHICH TRANSLATES THE MACHINE INDEPENDANT JANUS CODE INTO THE ASSEMBLY CODE FOR THE TARGET MACHINE, HAS TO BE REWRITTEN, I.E. ANOTHER SET OF MACROS FOR THE NEW MACHINE HAS TO BE DEFINED. THE NECESSARY DOCUMENTATION IS AVAILABLE FROM [FEI73].

THE PASCAL-COMPILER (WRITTEN IN PASCAL AND JANUS), THE STAGE2 MACROGENERATOR (DOS AND RSX-11 VERSION) [FEI74] AND THE PASCAL-11 USER'S GUIDE [FEI75] (20 PAGES) AS WELL AS THE WHOLE PASCAL-11 SYSTEM ARE AVAILABLE.

REFERENCES:

CDL73 CULLIAN,S.S., DOOLE,F.C. , WAITE,W.M.
THE MOBILE PROGRAMMING SYSTEM: JANUS
UNIVERSITY OF COLORADO, BOULDER, 1973
SOFTWARE PRACTICE AND EXPERIENCE

FEI75 FEIEREISEN,L.
PASCAL-11 USER'S GUIDE
UNIVERSITAET KARLSRUHE MAI-75

MEI74 MEIERHOF,H.M.
STAGE2 FOR THE PDP-11 CECOS 1974

PAS74 PASCAL COMPILER
AMMAN,U. , SCHILD,R. DATE:23/11/72
FACHGRUPPE COMPUTERWISSENSCHAFTEN
EIDG. TECHNISCHE HOCHSCHULE
CH-8006 ZUERICH

REVISION TO PRODUCE JANUS
LARRY B. WEBER
UNIVERSITY OF COLORADO SPRING 1973

REVISED BY LUCIEN FEIEREISEN
UNIVERSITY OF KARLSRUHE FALL 1974

WAI73 WAITE,W.M.
IMPLEMENTING SOFTWARE FOR NON-NUMERIC APPLICATIONS
PRENTICE-HALL IN., ENGLEWOOD CLIFFS, N.J. (1973)

WEB73 WEBER,L.B.
A MACHINE INDEPENDANT PASCAL COMPILER
MS-THESIS, UNIVERSITY OF COLORADO, BOULDER, 1973

I AM PLEASED TO BE OF ASSISTANCE, BOTH NOW AND IN THE FUTURE.

YOURS SINCERELY

LUCIEN FEIEREISEN

UNIVERSITÄT HAMBURG

INSTITUT FÜR INFORMATIK
Prof. Dr. H.-H. Nagel

Institut für Informatik
2 Hamburg 13, Schlüterstraße 66-72

Datum July 1st, 1975

Dear Mr. Richmond,

The enclosed summary informs you about our PASCAL-compilers available for the DECSystem-10. In case you enquired recently about our compiler and did not yet receive an answer, please excuse me. I have been busy (amongst other tasks) to prepare this version for distribution.

☐ You are on a distribution chain for three DECtapes and scheduled to receive them from

☐ Please check here if you are interested to obtain our PASCAL compiler.

Shipment requires (please check)

☐ 1 Dectape for the PASCAL-compiler generating directly executable, sharable object code

☐ 2 Dectapes for the PASREL-compiler generating LINK-10 compatible relocatable object code, the PASCAL source level debugging system (PASDDT), the crossreferencing program CROSS and the PASCAL-Help file (the latter one in German since I did not want to delay shipment any further by the time required to translate this file into English).

☐ 1 small MAGtape if you don't have Dectape drives at your installation. Since MAGtapes require more trouble at our site, DECtapes are preferred.
Please do not send tapes if you are located in continental US or Canada since I intend to refer your name to someone in your vicinity who has received these versions.

☐ Would you see a possibility to provide a copy of these files to someone else if asked to do so?

If you are interested, please return this questionaire to
H.-H. Nagel, Institut für Informatik
Schlüterstraße 70, D-2000 Hamburg 13

Yours sincerely

H.-H. Nagel

The PASCAL implementation for the DECSystem-10 has been considerably improved and enlarged. It now supports all phases in the generation, debugging and maintenance of PASCAL programs.

1. The editing phase by the formatting features of CROSS.
2. The compilation phase by offering:
   2.1 a compiler named PASREL generating relocatable object code compatible with LINK-10. Output from this compiler will automatically direct the loader to search the FORTRAN-library for standard functions SIN, COS etc. if necessary. External procedures (e.g. written in MACRO-10 or separately compiled PASCAL procedures) can be linked on. A source level debug option is available (see p. IV).
   2.2 a more compact, faster compiler named PASCAL generating directly executable, sharable object code. Use of this compiler is recommended, if

       no external procedures
       no debug option
       no standard functions from the FORTRAN library

   are required.
3. The debugging phase: breakpoints can be set at runtime based on source program line numbers. After a program stops at such a breakpoint, variable locations can be inspected and modified using the source program identifiers (see p. IV).
4. The maintenance phase: the program CROSS generates
   - an incentated source program listing with extension .CRL
   - markers in the left margin for each start and termination of nested statements
   - a crossreference list of all source program identifiers
   - a survey of the static procedure nesting
   - for each procedure a list of which procedures it activates and by which procedures it is activated
   - an indentated source program file with extension .NEW

The following section summarizes new features available in both compilers (see the PASCAL.HLP file distributed with PASREL for further information):

5.1 READLN {(<file identifier>)} skips over the rest of the current line until the next end-of-line is detected. It accepts further arguments.

5.2 PAGE {(<file identifier>)} appends a <carriage return><form feed> to the FILE OF CHAR denoted by <file identifier>. If none is given OUTPUT is assumed. <formfeed> advances to the beginning of the next page.

5.3 The procedures PACK and UNPACK have been implemented with an optional fourth argument, These procedures are much more effective in packing or unpacking larger arrays than a FOR-loop using indexed access to components of such an array.

5.4 The sequences (* and *) are recognized as opening and closing comment brackets in addition to % and \.

5.5 CROSS and both compilers accept the general file specification allowed by the TOPS-10 monitor.

5.6 A constant subrange may be given in sets, : e.g. ['A' .. 'C'] instead of ['A', 'B', 'C'].

5.7 An OTHERS branch may be specified in CASE-statements.

5.8 Compiler options (see also 5.14):

5.8.1 %$C+\ generates instructions for runtime checks at array indices and assignment to scalar and subrange variables.
%$C-\ suppresses code generation for runtime checks
Default: C+

5.8.2 %$L+\ appends the symbolic version of the object code generated to the source program listing for each procedure and adds the starting address of the object code for each source program line
%$L-\ no symbolic object code listing
Default: L-

Since runtime errors still give only the object code address besides the message identifying the error, compile the program with the compiler option L+ in addition to C+ in order to learn from this listing, to which source program line the error address belongs.

5.9 The following standard functions are available to both compilers

function	of result type	yielding
TIME	INTEGER	time in milliseconds
RUNTIME	INTEGER	CPU-time in milliseconds

5.10 For initialisation of global variables at compile time use INITPROCEDURE.

5.11 The LOOP statement is available.

5.12 The standard procedures RESET/REWRITE can be used with up to four optional arguments allowing full file specifications at runtime

5.13 Pascal programs to be compiled by PASREL may use the following additional standard functions (all functions and arguments are of type REAL) from the FORLIB on the logical device SYS:

SIN	COS	ARCTAN	EXP	SQRT	RANDOM
SIND	COSD	TANH	LN		
ARCSIN	ARCCOS		LOG		
SINH	COSH				

function	of result type	yielding
DATE	PACKED ARRAY[1..9] OF CHAR	'DD-MMM-YY' with D=day, M=month, Y=year

5.14 Following the head of a procedure/function declaration by
EXTERN <language symbol>;
will direct the compiler to provide for linkage to an external procedure/function.
<language symbol>:::=empty|FORTRAN|ALGOL|COBOL.
The language symbol determines the conventions for parameter passing to an external procedure/function. If none is provided, PASCAL is assumed. If any of the three nonempty language symbols is indicated, the loader is directed to search the corresponding library on logical device SYS.
If a group of PASCAL procedures without a main program has to be compiled separately, use %SM-\ at the beginning of the corresponding PASCAL source file. In this case the outermost procedure/function names will automatically be declared as ENTRY by the compiler. Include their .REL files when loading!

5.15 BREAK {(<file identifier>)} forces the current buffer contents to be output to the file specified by <file identifier>. If none is specified, TTY is assumed. This feature is useful, too, for intercomputer-communication at the PASCAL level.

To use the crossreference listing program

.RUN CROSS	
FILE: <filename>	after FILE: : is typed by CROSS give source filename in the format DEVICE: FILNAM.EXT [project#, prog#] everything except FILNAM may be omitted

To use PASCAL:

.RUN PASCAL	
*<filename>	source filename specification (see CROSS)
{NO} ERROR DETECTED	If an error has been detected, object code is not available for execution!
EXIT	
.RUN <filename> ##	The core requirement indicated by ## must be estimated from the length of LOW and SHR file plus space for stack and heap. Since error messages will appear if core space is insufficient, trial is often the quickest approach.

To use PASREL:

.RUN PASREL	
*<filename>	source file specification (see CROSS)
{NO} ERROR DETECTED	(see PASCAL)
HIGHSEG : M K	M indicates size of high segment (code) in Kwords
LOWSEG : N K	N indicates size of low segment (data) in Kwords
RUNTIME : T	T indicates CPU-time used for compilation
EXIT	
.LOAD <filename>	loading the program
LINK: LOADING	
EXIT	
.SAVE <filename> W	The total core requirement W must be given. W ≥ M + N + 4 Kwords If no debug option has been specified in the source program, the save-file can be made sharable by using the monitor command SSAVE <filename> W.
JOB SAVED	
.RUN <filename>	execute the program

Instructions to generate a new compiler version can be found at the beginning of each PASCAL-compiler source version.

Note:

Not yet implemented:

- formal procedure/function arguments
- branch out of a procedure/function (label declaration is not yet required)

6. DEBUG option

6.1 Indicate debug option in (part of the) source program text: %D+\. If no debugging is required in later parts, follow the section to be debugged by %D-\ since this will save core and runtime in those sections not to be debugged.

6.2 Use PASREL etc. (see above) to generate an executable SAV file, giving W ≥ M + N + 8 to allow working space for the debug code.

6.3 Get the listing of the compiled program in order to know exactly where to set breakpoints by
PRINT <filename>.LST

6.4 .RUN <filename>	begin execution of this program
*	answer with <carriage return>
$ STOP AT MAIN BEGIN	
$	enter breakpoints using the following commands
6.5 $ STOP <LINE>	set a stop at the begin of the source line indicated by line number <LINE>
	<LINE> ::= <LINENUMBER> | <LINENUMBER>/<PAGENUMBER>
	<LINENUMBER>::= <UNSIGNED INTEGER>
	<PAGENUMBER>::= <UNSIGNED INTEGER>
	If no pagenumber is given, 1 is assumed.
6.6 $ STOP NOT <LINE>	delete breakpoint at line <LINE>
6.7 $ STOP LIST	list all current breakpoints on the terminal
6.8 $ <VARIABLE> =	give the current contents of the location indicated by the source identifier (possibly expanded by qualifiers); the scope rules applying to the source statement corresponding to the current breakpoint uniquely determine the variable location from the identifier given. All qualifiers legal in Pascal may be used (i.e. pointers, record fields, array components)
6.9 $ <VARIABLE>:=<VARIABLE OR CONSTANT>	The variable or constant value on the right hand is assigned as current value to the variable indicated in the left side
6.10 $ TRACE	Backtrace of procedure nesting from Breakpoint to main; exposes activating procedures with linenumber/pagenumber of activation points
6.11 $ END	leave debug mode and continue execution. If any breakpoint is reached, the message
6.12 $ STOP AT <LINE>	appears on the terminal
6.13 asynchronous stop	If the program has been compiled with the debug option, it's execution can be interrupted by two successive control C:
↑C ↑C	typing DDT will then transfer control to the debugging mode
.DDT	
$ STOP BETWEEN <LINE1> AND <LINE2>	will appear on the terminal to indicate where the program has been interrupted. Use of commands described in 6.5 through 6.11 is now possible.

INSTITUT NATIONAL POLYTECHNIQUE
DE LORRAINE
DÉPARTEMENT INFORMATIQUE

Nancy, le 4 Juillet 1975

Monsieur Alain TISSERANT
Ecole des Mines
Département Informatique
Parc de Saurupt
54042 NANCY CEDEX
France

Mr George H. RICHMOND
Pascal Newsletter Editor
University of Colorado
Computing Center
3645 Marine Street
BOULDER, Colorado 80302
U.S.A.

Dear Sir,

A Pascal compiler for Télémécanique T1600 and Solar minicomputers is under development; a first version will be available in September 1976. These computers have a 16 bits words size, and no virtual memory facility. Our compiler will run with 24K words.

We are implementing a segmentation mechanism, reflecting both Pascal programs structure and the Solar computer architecture. At each procedure call, a new "segment" will be created for code and local data. A specialised monitor manages core memory, and swap operations.

The Pascal-P compiler is being modified (without change in the language accepted), in order to get code adapted to our data structures representation and our particular procedure linkage method.

We are using an existing Pascal compiler (on the CII Iris 80) for first binary code generation of the Solar compiler.

All these mechanisms are fully transparent to the user. By careful use of the particularities of special instructions and the architecture of the computer, we hope to get a high speed, easy to use Pascal system.

Sincerely,

A. TISSERANT

11

THE UNIVERSITY OF MANITOBA
DEPARTMENT OF COMPUTER SCIENCE
WINNIPEG, CANADA R3T 2N2

WBF/CMcL

15th July, 1975.

George H. Richmond,
Computing Center,
University of Colorado,
3645 Marine Street,
BOULDER,
Colorado 80302,
U.S.A.

Dear Mr. Richmond,

The enclosed is being sent to all the people who have written to us requesting information about our PASCAL implementation.

I realize that up until now, very little information about the project has been released. I think that this description gives a fair representation of our compiler as it currently exists.

The compiler was written as my Ph.D. project under the supervision of Professor James M. Wells. Professor Wells is currently on sabbatical leave in Ottawa. For this reason, I would appreciate it if you would include my name on your PASCAL Newsletter distribution list.

I am very interested in descriptions of other PASCAL Compilers and interpreters for IBM machines. If you have any information on core requirements, compile speeds, and whether or not the full language is supported, for the Grenoble, Stanford or Cambridge projects, I would appreciate hearing from you.

Yours sincerely,

W. Bruce Foulkes

(Enc.)

12

THE UNIVERSITY OF MANITOBA

DEPARTMENT OF COMPUTER SCIENCE WINNIPEG, CANADA R3T 2N2

WBF/CMcL July, 1975.

Dear Sir or Madam,

We are announcing the availability of a PASCAL compiler for IBM 360/370 computers, developed by the Department of Computer Science at the University of Manitoba. The compiler was written by Mr. W. Bruce Foulkes under the supervision of Professor James M. Wells.

The compiler is one-pass and uses a top-down parsing strategy. A generated assembler parser is produced by the translator writing system SYNTICS. All semantic routines are written in PL360, while system interfaces are written in assembler.

The compiler is not a re-write, modification, or bootstrap of any previous PASCAL compiler. The compiler uses some routines provided by the SYNTICS system and borrows some ideas and code from the ALGOLW compiler for code generation, built-in functions, and I/O.

This version of the compiler requires approximately 170K bytes. This size is variable, but the minimum size for compiling a meaningful program is approximately 150K.

Compile speed for test programs has been in the range 125-200 lines per second on an IBM 370/158. This excludes the set-up time of approximately 0.4 sec.

A great deal of compile-time checking is done and approximately 130 different error and warning messages are provided.

The production of run-time checking code for array subscripts, subrange assignments, values returned by PRED and SUCC, etc., can be turned on or off at will. Run-time interrupts are trapped with a SPIE macro. There are about 40 run-time error diagnostics in total. Each error diagnostic consists of an error message, location in the current segment, the invalid value if

.../2

appropriate, and a traceback of all segments invoked.

Linkage, although not completely standard between PASCAL segments, appears to be standard IBM to any external segments, allowing linkage to routines written in other languages.

The compiler supports a subset of the language described in the Revised Report. The main omissions are the following:

- only the standard input and output files SYSIN and SYSPRINT are supported. All I/O is done through the use of READ, READLN, WRITE, WRITELN, EOLN, and EOF. The I/O is not exactly standard; in particular, formatting is also allowed on input.
- the program header is not used. SYSIN and SYSPRINT must always be provided.
- packed arrays and records are not supported.
- only the simple forms of procedures NEW and DISPOSE are allowed. Tagfield values may not be specified. No garbage collection is done.
- global labels are not implemented.
- subranges of characters are not allowed.

With the above exceptions, the language supported is very close to that described in the Revised Report.

Seven standard scalar types are provided: SHORT INTEGER, INTEGER, REAL, LONG REAL, BOOLEAN, CHAR and STRING.

Built-in functions include: ABS, SQRT, EXP, LN, LOG, SIN, COS, ARCTAN, SQR, SUCC, PRED, ODD, ROUND, TRUNC, ORD, CHR, CARD and CPUTIME.

The compiler checks for overflows on all tables and produces terminal error messages. The main table sizes may be modified using parameters on the EXEC card. The source for an initialization routine will be provided which sets the size limits for all compile-time tables, and also sets defaults for compiler flags (such as whether run-time checking code should be produced). This should allow the compiler to be tailored to suit the needs of any installation. The remainder of the source will not be released at this time.

There are two main limitations imposed by the compiler. The maximum nest allowed for procedure and function declarations is 5, and all program segments are restricted to 4K bytes of code.

The compiler has not undergone large-scale production testing; for this reason, no guarantees are made as to its reliability. Considering the interest which has been shown in the compiler, we feel that we cannot justify delaying its release any longer.

.../3

The PASCAL compiler may be acquired by sending the PASCAL Order Form and the signed DISTRIBUTION AGREEMENT together with $50 (payable to the Department of Computer Science, University of Manitoba), to the PASCAL Distribution Manager.

The tape will contain the object modules necessary to generate the compiler along with sample JCL, test programs and a user's guide.

After a suitable test period an updated version of the compiler may be offered, but no promises to this effect are made.

We hope that few problems will be encountered.

Please see the enclosed material if you are interested in ordering the compiler.

W. Bruce Forsha

PASCAL Distribution Manager

UNIVERSITY OF MINNESOTA
TWIN CITIES

University Computer Center
227 Experimental Engineering Building
Minneapolis, Minnesota 55455

July 16, 1975

To the Editor, Pascal Newsletter:

We at the University of Minnesota would like to participate in a Pascal User's Group for North America to help distribute and support PASCAL. In communications with various other sites: Alfred Towell at Indiana University, Dave Tarabar at University of Massachusetts, and George Richmond at Colorado University, there seems to be a desire to form such an association. Perhaps a kind of conference would be appropriate for getting started.

The usage of PASCAL at our site has been heavy lately. PASCAL is being used in general applications on not only the MERITSS CDC6400 state-wide timesharing network but also at the University of Minnesota's batch computing facility, a CDC CYBER 74. We have locally modified the PASCAL system totally in a cosmetic way: fixing bugs, making interface changes for the KRONOS 2.1 operating system and making PASCAL available under the TELEX timesharing supervisor available with KRONOS. A dozen or so sites have our modifications for interactive usage, however we don't know as yet how useful they are for INTERCOM under SCOPE 3.4, although Urs Ammann of Zurich seems to think we went about our changes in a decent way, which may make them a good model to follow.

We would like to caution others about what "improvements" they make to their implementation of standard PASCAL.* For one thing the old compile to core PASCAL compiler for CDC machines was changed by many people in ways that violated the underlying principles of the language. For example: simplicity in design (which implies simplicity in description-in other words, few exceptions to the rules); a specific infraction being the addition of a step specification in for loops by one installation.

In reply to a letter to the editor of 6 August, 1974 by George Poonen in Pascal Newsletter No. 3, we would like to reply that we also deplore "dialect_s" of PASCAL. However, BLAISE and SUE are not dialects but other PASCAL-like languages. Further, the Axiomatic Definition and the Revised Report define the standard semantics and syntax of the language.

* See the very interesting article: "An Assessment of the Programming Language PASCAL" by Niklaus Wirth in the June, 1975 issue of SIGPLAN Notices, Proceedings: International Conference on Reliable Software.

If what is meant is trying to resolve a standard for extensions to the language (such as a value-initialization facility) then that is another question. Perhaps this should be investigated.

To have PASCAL succeed at a given installation (with the goal of being used as much or more than FORTRAN) may require the local maintainer's recognition of his or her resonsibility to and power over the lives of all the PASCAL users affected. Also consider the ideas put forth by Prof. William Waite, Colorado University, in a guest editorial to the Vol. 3, No. 3 1973 issue of Software Practice and Experience. He uses the analogy of organisms (language processors) in an ecosystem (computer center). PASCAL is a very good product which sells itself-but by giving the compiler inadequate support, it can fail with certainty. Support includes not only simple availability, but also publicity and all the other amenities of programming life which now make FORTRAN easy to use. Examples are utility routines, libraries, program preparation equipment with the proper character sets, etc.

This summer we are engaged in additional enhancements to PASCAL's support at the University of Minnesota. The Computer Science Department here has now adopted PASCAL throughout its curriculum. By October we will be willing to share with other sites several of the documents we will have produced.

Andrew Mickel John Strait

Andrew Mickel and John Strait
University Computer Center
227 Experimental Engineering
University of Minnesota
Minneapolis, MN 55455

17

AM/JS/ks

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

Mr. George H. Richmond
University of Colorado
3645 Marine Street
Boulder, Colorado 80302
U.S.A.

July 25, 1975
JSM/HG

Dear Mr. Richmond.

Having received your list of PASCAL implementations I will ask you to correct the information of our compiler thus:

Implementation Route : PASCAL-P1 Bootstrap
Implementation Status: Complete, Available for distribution.

Sincerely

J. Steensgaard-Madsen

18

encl.

PRELIMINARY DESCRIPTION OF A PASCAL COMPILER FOR UNIVAC 1100.

J. Steensgaard-Madsen
Datalogisk Institut
Sigurdsgade 41
DK-2200 Copenhagen
DENMARK

Introduction.

The following text describes in short a PASCAL compiler for UNIVAC 1100 machines operating with EXEC 8. The system is developed from a PASCAL P compiler obtained from professor Niklaus Wirth. The work has been done at Datalogisk Institut, University of Copenhagen by three students

Arne Kjær
Jan Højlund Nielsen
Henrik Snog

and a teacher

Jørgen Steensgaard-Madsen

The present (preliminary) description is not complete in every detail, but tries to convey all relevant information to the vast majority of users. It should be used together with the book

PASCAL User Manual and Report
by Kathleen Jensen and Niklaus Wirth
Springer Verlag 1974
(Lecture Notes in Computer Science no. 18).

Representation of PASCAL programs.

The representation of PASCAL programs for UNIVAC 1100 is based on the standard representation using ASCII character set. This is converted to FIELDATA using the rules fixed by UNIVAC, except for opening and closing brace, { and } , which are not used. This means that ↑ is converted to Δ and that comments are enclosed in (* and *).

Restrictions. (July 75)

Variables of type TEXT, except INPUT and OUTPUT, cannot be used. Page procedure is not implemented.

DISPOSE is not implemented.
File components containing files cannot be used.
Standard procedures cannot be passed as parameters.
Fields of packed structures cannot be substituted for var parameters.
Sets must be over base types containing at most 72 values (in case of INTEGER it must be a subrange contained in 0 .. 71).

Additional standard identifiers.

```
const     ALFALENG = 12;
type      ALFA = PACKED ARRAY [ 1 .. ALFALENG ] OF CHAR;
         HALFA = PACKED ARRAY [ 1 .. 6 ] OF CHAR;

procedure HALT;
   (* terminate execution *)

procedure MARK ( var I : INTEGER );
   (* returns with I information to be used in recollecting
      storage allocated by subsequent calls of NEW *)

procedure RELEASE ( I: INTEGER );
   (* releases storage allocated by calls of NEW since
      the call of MARK that must have set I *)

procedure WRITEPAGE;
   (* advances the printer so that next line is printed
      as fist line on a new page *)

procedure CLOSE ( var F : any file );
   (* this is a file operation, which must be executed
      as the final operation on external files *)
```

Input / Output.

The procedures read and readln take as parameters variables of type CHAR, INTEGER, REAL, HALFA and ALFA. Except in case of CHAR, where just one character is read, leading blanks are skipped and the following characters are analysed. In case of ALFA (HALFA) at most 12 (6) nonblank characters are read and stored left justified and blankfilled. With multiple parameters an error exit caused by an end of file condition will only occur if EOF is TRUE prior to the call.

Parameter specifications.

A name may be associated with a specification of formal parameters. This is done in the parameter definition part placed after the variable declaration part. The syntax is

```
<parameter definition part> ::=
  param  <parameter declaration> ;
        {<parameter declaration> ;}

<parameter declaration> ::= <parameter identifier> =
  ( <formal section> {; <formal section>} )
```

The parameter names may then be used in the declaration of procedures and functions

```
<procedure heading> ::=
  procedure <identifier> ; |
  procedure <identifier> ( <specification> );

<function heading> ::=
  function <identifier> : <result type> ; |
  function <identifier> ( <specification> ): <result type>;

<specification> ::=
  param <parameter identifier> |
  <formal section> {; <formal section>}

<formal section> ::= <formal parameter section> |
  procedure <identifier> ( <parameter identifier> ) |
  function  <identifier> ( <parameter identifier> ) :
  <result type>
```

This syntax allowes the complete specification of formal procedures and function which is required in PASCAL for UNIVAC 1100.

Loop statement.

For historical reasons the loop statement is included in PASCAL for UNIVAC 1100.

```
<loop statement> ::=
  loop {<statement> ;}
  exit if <expression> {; <statement>}
  end
```

The program scheme

```
  loop P1; exit if B1; P2 end
```

is equivalent to

```
  P1; while not B1 do begin P2; P1 end;
```

Case statement.

In the case statement case labels may be specified by subranges in usual notation. The final end in a case statement may be replaced by otherwise <statement> meaning that the statement following otherwise will be executed if none of the labelled statements is selected for execution.

TECHNISCHE HOGESCHOOL TWENTE

ONDERAFDELING DER TOEGEPASTE WISKUNDE

Mr. G.H. Richmond
University of Colorado
Boulder Colorado 80302
Computer Center

ONDERWERP: KENMERK: TW75/INF/302 ENSCHEDE, 11 augustus 1975

Dear Mr. Richmond,

Regarding the status of the PASCAL-implementation for the PDP11 series.

Date: 8 august 1975

Implementation Route: PASCAL-P1, Cross Compiler described in PASCAL to be run on PASCAL system for DEC-10.

Target Machine : PDP11 series, no O.S. requirements (all models).

Implementation Status: Testphase nearing completion. Available for Distribution by Dec. 1975.

Restrictions : except for standardfiles INPUT & OUPUT, files are not implemented. Jump out of procedure not implemented.

Extensions : formal/procedure/function specification required. Array-parameters with unspecified bounds are allowed. Functions may deliver results of any type.

Yours sincerely,
C. Bron
Drs. C. Bron.

ENSCHEDE - DRIENERLO - POSTBUS 217 - TELEFOON: 05420 - 99111 - TELEX 44200

BASSER DEPARTMENT OF COMPUTER SCIENCE
School of Physics (Building A28),
University of Sydney, N.S.W. 2006

15th August, 1975

Mr. G.H. Richmond,
University of Colorado,
Computing Centre,
3645 Marine St.,
Boulder, Colorado,
U.S.A. 80302

Dear Mr. Richmond,

In response to your circular of June 5, I am providing the following information.

As Dr. Sedgwick recently left the Department in order to take up a position in Toronto, the contact for the local Pascal-P2 implementation is myself. The status of the implementation is "progressing" with completion anticipated around the end of this year. The main hold-up has been the lack of documentation for the B1726's operating system.

Since the B1726 is user-microprogrammable and bit-addressable, the implementation strategy is basically that of microcoding the Pascal-P interpreter. (In fact, all languages on the B1726 are implemented in this manner.) As well, the compiler has been modified so that it supports the EBCDIC character set and generates "machine" instructions which support bit-addressable data items of arbitrary length. Our configuration parameters have been chosen as follows: 16777215, 25, 34, 8, 1, 72, 24, 16, 24. Please note that the unit of storage is obviously the bit, and that the setsize of 72 is arbitrary and will be extended to 256 eventually, to allow sets of chars. Another consequence, of bit-addressability is that the packed keyword becomes superfluous, thus eliminating one "restriction" in the present compiler.

It is also encouraging to observe that initial estimates of the compiler's size place it well below that of the Burroughs-supplied compilers (including Fortran, Cobol, RPG), with the exception of the Basic compiler.

Yours sincerely,

Antony J Gerber

Antony J. Gerber

University of Florida
Gainesville, 32611

Intercollege Department of
COMPUTER AND INFORMATION SCIENCES
512 Weil Hall
904-392-2371

August 18, 1975

Degree Programs in the Colleges of
ARTS AND SCIENCES,
BUSINESS ADMINISTRATION
and ENGINEERING

Mr. George H. Richmond
University of Colorado
Computing Center
3645 Marine Street
Boulder, Colorado 80302

Dear Mr. Richmond:

As I mentioned in our telephone conversation last week, I have successfully transported the PASCAL 'P' compiler to the TI 980A, a 16-bit word minicomputer.

Machine requirements are:

1. Minimum of 36K of main memory (4K for operating system)
2. disk
3. Silent 700 console with dual cassettes .

The P-code for the compiler occupies 27,936 words (2 words/instruction) and the loader-interpreter occupies 3,744 words. Storage in main memory is dynamically allocated so the system can be run on any machine having at least 36K of main memory.

Since the P-code for the compiler is so large (approximately 17,000 records) it was physically split into three parts on a large computer and transmitted across telephone lines to cassettes. Then it was merged back into one file on the TI disk.

Pertinent configuration parameters are:

INTSIZE = 1	SETSIZE = 4
REALSIZE = 2	PTRSIZE = 1
CHARSIZE = 1	STRGLGTH = 6
BOOLSIZE = 1	INTBITS = 15

Sincerely yours,

Gilbert J. Hansen

Gilbert J. Hansen
Assistant Professor

The University of Tasmania

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

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

Information Science Department.

IN REPLY PLEASE QUOTE:

FILE NO. .....

IF TELEPHONING OR CALLING

ASK FOR .....

18th August, 1975.

Mr. G.H. Richmond,
Computing Center,
University of Colorado,
BOULDER, COLORADO.

Dear Mr. Richmond,

In response to your letter relating to communication between Pascal implementors, I can only heartily agree. I therefore give you the following points relating to our work on Pascal-P2.

Status: Incomplete (1975 August)

Purpose of implementation: Investigation of portability; use of Pascal in teaching.

Route: Piggybacking via PASCAL-1 on Cyber 72 to produce Burroughs B6700 'assembly code', thence 'assembly code' to be processed for route: This two-step process separates the easy part (generating B6700 code) from the hard part (getting the B6700 to accept it).

Impression of package: Far too little thought given to portability and to documentation. Pascal-P2 still betrays its heritage as a CDC-biassed product in subtle but annoying ways, though vastly better than earlier Pascals; it remains a test-bed product designed for a restricted purpose, and despite the claims made for it in the documentation, could be made vastly easier to port (whether bootstrapped or piggybacked).

Main implementation difficulties: Doing sensible things on a highly structured computer (without a linear von-Neumann memory) with multiple-word objects, particularly allocated in the heap. It seems a great pity to have to forego the many advantages of the B6700 architecture because of some of Pascal's features, and yet trying to use all the good features may well lead to excessive memory fragmentation and segmentation; also possibly complex code generation for different handling of constructs. For example should all *records* be individual *segments?* or should a simulated linear store (= a large declared vector) be used to pretend to be a more conventional machine?

Useful information for other implementors: A few sheets showing the (static) frequency of occurrence of each of the SC-machine instructions is available on request, also doublet frequencies. This shows which instructions may be ignored or simplified in interpreting or macro-expanding the code, and where most of the space goes (and very likely the time too). Since this route was discarded as unnecessarily difficult, no dynamic execution frequencies are available.

Also note that sets of 48 bits are sufficient to bootstrap up the Pascal-P compiler itself (the largest set has 48 elements). The statement in the documentation relating to 59 bit sets is simply a CDC hangover which has not been checked. This has significance to 48-bit machines (as B6700).

Likely completion date: November/December 1975.

Yours sincerely,

A.H.J. Sale

A.H.J. SALE,
Professor of Information Science.

UNIVERSITY OF CALIFORNIA, SAN DIEGO

DEPARTMENT OF APPLIED PHYSICS AND INFORMATION SCIENCE
COMPUTER SCIENCE DIVISION, C-014

LA JOLLA, CALIFORNIA 92093

August 22, 1975

Professor A.H.J. Sale
University of Tasmania
Box 252C, G.P.O.
Hobart,
Tasmania 7001

Dear Professor Sale:

We are indeed working with PASCAL on the B6700. Whether the work is of immediate interest to you is another question. Making PASCAL into a stable B6700 product for users is a secondary objective of our project. Our primary aim is to create an interactive student debugging environment on the PDP-11, with virtually all of the software written in PASCAL.

The overall objectives of the project are described in the enclosed project prospectus. Students will interact with PASCAL on the small machines in a manner very similar to the debugging environment of APL on IBM 360/370 systems. PASCAL is interpreted using a modified version of the Zurich P-Machine recently released. The main purpose of the modifications is to reduce the size of the compiled code so that the PASCAL compiler can fit within the limited core of a small machine. Yes we have done the same kinds of statistical studies represented in the reports you kindly sent, though our data is not in as elegant a form. We are confident that the compiler can be run on a PDP11 with at least 20K words of memory. We are hoping to reduce that amount further to perhaps 16K, when time permits. Currently the interpreter is operating, but has yet to be tried with the whole compiler on the PDP-11.

We are using the modified PASCAL compiler on the B6700 as a tool for developing the new PASCAL system, and generating pseudo-code for the PDP-11. Having started with an interpreter for the Zurich P-Machine, we have progressed through various stages of bootstrapping to get a system compatible with the PDP-11 objective, and the interactive system objective. Concurrent with the work using the interpreter, we also have an advanced student programmer writing an assembler which converts the compiler P-code output into directly executable B6700 code. The B6700 code has been executed with small programs, and should be running the whole expanded compiler within a week or so. This compile-assemble system manages its memory in one large array in a fashion similar to that used on conventional machines. We are using the B6700 SWAPPER for much of our batch work, and hence have been able to use DIRECT (non-overlayable) array space for this purpose to enhance the speed of the processing.

The short-term objective for the B6700 compile-assemble system is to provide a back-up means for students to use for PASCAL homework problems starting in late September. Our PDP-11 equipment is not all here yet, and we clearly will not be ready to use the small machines with students during the first few weeks of the Fall Quarter. Over the time period of the academic year about to start, we will almost certainly have someone complete the job of making a PASCAL compiler that can generate B6700 code directly. Yet to be resolved is the question of whether we can map the PASCAL data structures into the array-row structures of the B6700 without doing violence to the basic approach of the P-compiler.

The interpretive system is slow on the B6700, as might be expected. The major consumer of time is the low level character processing in the INSYMBOL and NEXTCH procedures. We have changed these procedures completely, so as to depend upon installation intrinsic functions (Standard Procedures) that make use of the B6700 string processing hardware. The GETSYM intrinsic returns information on each successive token in an area of stack that serves as a scanner information block. This provides a clean interface between compiler and interpreter, but it runs about half as fast as an earlier less-clean version (part way through the bootstrapping) in which virtually all of the work of INSYMBOL was done in an ALGOL intrinsic. The current B6700 interpretive version takes about 10 minutes of processor time to compile the source file from Zurich. We expect the compile-assemble version, and also the PDP-11 version, to run roughly five times faster than that.

During the next two months we will be up to our ears in alligators getting this system completed well enough to use for teaching. At a later stage, I would be happy to share more details with you.

Sincerely,

Ken Bowles

Kenneth L. Bowles
Professor (Computer Science)

The University of Tasmania

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

Telephone: 23 0551. Cables "Tasuni" Telex: 58150

Information Science Dept.

9th September, 1975.

Mr. Richmond,
Computing Centre,
University of Colorado,
3645 Marine St.,
BOULDER, COLO.

Dear Mr. Richmond,

Pascal-P Documentation

I have received, in response to some of my correspondence, a copy of some error notes on Pascal-P detected by the University of Karlsruhe. The documentation is in German, and I have attempted to translate the sense of the notes with the results attached. I hope that this may be of use to other Pascal-P implementors. I have asked the originators to see if my translation accords with what they thought they said in case I have missed some idiomatic nuance (technical German terms are quite as mystifying as English ones until decoded; witness "bugs") and I shall let you know if there are any alterations or additions.

Yours sincerely,

A.H.J. SALE,
Professor of Information Science.

Encl.

LIST OF PASCAL DEFECTS

The following errors and defects were found during the implementation of the PASCAL system.

1. Defects in the compiler which adversely affect the bootstrapping to different computers.

   (a) Assumptions are made about the collating sequence of the character set which are neither warranted nor defined in the axioms concerning character type. In particular:

   - The translator assumes that the characters '+' and ';' enclose all operators in the declarations of

     ```
     SSY, SOP:  array ['+';..';']
     ```

   - The test which determines whether a character is alphanumeric is formulated as

     ```
     ORD(CH)  >= ORD('A') AND ORD(CH)  <= ORD('9')
     ```

   (b) The compiler accumulates errors only during the translation of a line of source text. At the end of the translation the compiler cannot determine automatically whether the generated code is correct, or whether the PASCAL program was in error or not.

   (c) The constant CHARSIZE, which is used to parameterize the compiler for various computers, is employed with the meaning "Storage units per character" and not "characters per storage unit". While this is clearly set out in the documentation it is unexpected.

2. Deliberate restrictions imposed by the compiler.

   (a) Only the first 8 characters of identifiers are significant in distinguishing them; remaining characters are ignored.

   (b) String constants are limited to a maximum of 16 characters.

3. Errors in the compiler.

   (a) When translating a PASCAL program which does not have the terminating symbol *END*. the compiler hangs in an infinite loop. The error is in procedure NEXTCH. It has the structure:

```
PROCEDURE NEXTCH;
BEGIN
    IF EOL THEN BEGIN ... END;
    IF NOT EOF(INPUT) THEN
    BEGIN ..."read next symbol and assign value to global"
    END ELSE WRITE(OUTPUT,'EOF ENCOUNTERED')
END;
```

In this case NEXTCH is called by SKIP (via INSYMBOL) until the terminating symbol is read. If NEXTCH in the compiler is not altered, the compiler loops endlessly. An error exit in NEXTCH would solve the problem but this is not permitted in PASCAL-P. (The error was probably not noticed in the CDC-implementation as multiple calls to EOF with the value TRUE cause a program-dump in that system. This meaning of EOF is not endorsed in the PASCAL definition and is not obvious.)

(b) When in the procedure INSYMBOL a symbol is read which does not belong to the recognised symbols, an error is signalled but the next symbol is not read in. When SKIP initiates the consequent search to recover from the error, the compiler finds itself in a loop, as subsequent calls continue to find the undefined symbol.

(c) The variable EOL should be initialised to the value FALSE (not TRUE), since otherwise the first action of the compiler is to issue a read command, and the counting of source lines starts at two.

4. Aesthetic defects in the compiler.

(a) To comply with PASCAL-P, the LOOP-EXIT construction has had to be deformed into a REPEAT-UNTIL construction. The change was made with too narrow a view, with the result that the constructions are much more difficult to understand. The loop construction, which is to be found in most procedures, has for example the following form (taken from INSYMBOL):

```
REPEAT
    WHILE CH = ' ' AND NOT EOL DO NEXTCH;
    TEST:=EOL;
    IF TEST THEN NEXTCH
UNTIL NOT TEST;
```

(b) There is an error message which is emitted not only by error number 117, but also with the message *'TYPE-ID' <identifier>*. This message does not match the source line and the following error-messages (and their lines) in the procedure ENDOFLINE, but seems to be issued immediately when the error is recognised in the syntactic and semantic analysis. The cause has not been found despite an intensive search. Thereafter the layout of the program listing, error numbers, and error text becomes unrecognisable.

5. Properties of the compiler which affect interpretation.

(a) The compiler produces the same commands as for all other objects for packed arrays of characters (strings). Also at the level of the interpreter strings must be treated as arrays of single characters.

(b) Parameterising the compiler for the storesize of sets (SETSIZE=2) does not result in efficient store utilisation in the interpreter. The formal parameters of procedures are set up with the maximum size of objects of type INTEGER, REAL, CHAR and SET. Similarly all load and store commands ignore the type of the data-types and are not parameterised. As a consequence if the stack is not to be full of objects all of the largest size, the load and store instructions have to inspect type-tages in the interpretive store.

Keeping type-tags (as implied by a direct interpreter implementation) is incredibly wasteful of store, and isufficient thought has been given to the problems of wordsize in claiming portability.

(c) The convention of the interpreter is not followed when code is generated for formatted output of strings. For example:

WRITE(OUTPUT,'ABC':10) is compiled to:

```
LCA 'ABC'
LDCI  10
LDCI   3
LAO    5
CSP  WRC
```

The interpreter expects the code:

```
LCA 'ABC'
LDCI   3
LDCI  10
LAO    5
CSP  WRC
```

NOTE: Translated from the original GERMAN supplied by the University of Karlsruhe (1975 August 21), with free translation.

1975 September 1
A.H.J. Sale,
Department of Information Science,
University of Tasmania,
G.P.O. Box 252C,
HOBART, TASMANIA. 7001