

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

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 := ft ; get(f) , and
write (f,x) to be synonymous to the sequence
ft := 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 := ft ; get(f) and
and
writeplop (f : file of plop ; x : plop);
begin ft := 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

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

Germany

KARLSRUHE, 30-JUN-75

MR. GEORGE H. RICHMOND
UNIVERSITY OF COLLEGE
COMPUTING CENTER
FSR 43
COLLEGE, COLLEGE 80302

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 $0 \leq x < 1$, it is not guaranteed that $1 \leq x * 10 < 10$, and therefore $\text{trunc}(10 * x)$ 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.

3

DEAR MR. RICHMOND,

I THANK FOR YOUR LETTER FROM THE 16-MAY-75. THE PASCAL COMPILER BASED ON JANUS HAS BEEN IMPLEMENTED ON THE FDP-11/45 RUNNING UNDER THE CCS/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 (CCL73).

THE MACROGENERATOR STAGE2 (MAI73, MEI74) MAPS THIS SYMBOLIC JANUS CODE INTO THE POP-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 POP-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 424 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 FDP-11, WHEREAS THE EXECUTION SPEED IS ABOUT THE FACTOR 2.65.

ALL FEATURES OF THE USED PASCAL LANGUAGE (CLASS AND ALPHA 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 (ALPHA TYPE). THE I/O CONCEPT INCLUDES CONCURRENCY AND EXPLICIT 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 FDP-11/45.

4

THE PASCAL-COMPIILER CAN BE MOVED TO ANOTHER COMPUTER WITH A LIMITED AMOUNT OF EFFORTS: 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 MACHINES FOR THE NEW MACHINE HAS TO BE DEFINED. THE NECESSARY DOCUMENTATION IS AVAILABLE FROM (HE873).

THE PASCAL-COMPIILER (WRITTEN IN PASCAL AND JANUS), THE STAGE2 MACROGENERATOR (CCS AND RSX-11 VERSION) (HE174) AND THE PASCAL-11 USER'S GUIDE (HE175) (22 PAGES) AS WELL AS THE WHOLE PASCAL-11 SYSTEM ARE AVAILABLE.

REFERENCES:

- CC173 GULLIAN, S.S., POCLE, F.C., WAITE, W.M.
THE MOBILE PROGRAMMING SYSTEMS JANUS
UNIVERSITY OF COLORADO, BOULDER, 1973
SOFTWARE PRACTICE AND EXPERIENCE
- HE175 FEIEREISEN, L.
PASCAL-11 USER'S GUIDE
UNIVERSITÄT KARLSRUHE MAI-75
- HE174 HEINRICH, F.H.
STAGE2 FOR THE PDP-11 CECLS 1974
- PAS74 PASCAL COMPILER
AMMAN, U., SCHILD, R. DATE: 23/11/72
FACHGRUPPE COMPUTERWISSENSCHAFTEN
EING. TECHNISCHE HOCHSCHULE
CH-8006 ZÜRICH
- REVISION TO PRODUCE JANUS
LARRY B. WEBER
UNIVERSITY OF COLORADO SPRING 1973
- REVISED BY LUCIEN FEIEREISEN
UNIVERSITY OF KARLSRUHE FALL 1974
- WA173 WAITE, W.M.
IMPLEMENTING SOFTWARE FOR NON-NUMERIC APPLICATIONS
PERTICE-HALL INC., ENGLEWOOD CLIFFS, N.J. (1973)
- WE173 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
3 Hamburg 10, Schillerstraße 66-72

INSTITUT FÜR
INFORMATIK

Prof. Dr. H.-H. Nagel

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

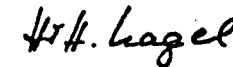
2 Dectapes for the PASREL-compiler generating LINK-10 compatible relocatable object code, the PASCAL source level debugging system (PASDDT), the crossreferencing program CROSS and the PASCAL-Help file (the latter one in German since I did not want to delay shipment any further by the time required to 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 questionnaire to
H.-H. Nagel, Institut für Informatik
Schlüterstraße 70, D-2000 Hamburg 13

Yours sincerely



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 deburring 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 indented 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 indented 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 %%L-\ 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 %%M-\ 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.

III

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#, progr.#]
 everything except FILNAM may be omitted

To use PASCAL:

```
.RUN PASCAL
* <filename>
[NO] ERROR DETECTED
```

source filename specification (see CROSS)
 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>
[NO] ERROR DETECTED
HIGHSEG : M K
LOWSEG : N K
RUNTIME : T
EXIT
```

source file specification (see CROSS) (see PASCAL)
 N indicates size of high segment (code) in Kwords
 N indicates size of low segment (data) in Kwords
 T indicates CPU-time used for compilation

```
.LOAD <filename>
LINK: LOADING
EXIT
.SAVE <filename> W
```

loading the program
 The total core requirement W must be given. $W \geq M + N + 4$ 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)

IV

6. DEBUG option

6.1 Indicate debug option in (part of the) source program text: $\$D\backslash$. If no debugging is required in later parts, follow the section to be debugged by $\$D\backslash$ 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 $W \geq M + N + 8$ 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>
 *
 \$ STOP AT MAIN BEGIN
 \$

begin execution of this program
 answer with <carriage return>

6.5 \$ STOP <LINE>

enter breakpoints using the following commands
 set a stop at the begin of the source line indicated by line number <LINE>
 <LINE> ::= <LINENUMBER>!
 <LINE> ::= <LINENUMBER> / <PAGENUMBER>

<LINENUMBER> ::= <UNSIGNED INTEGER>
 <PAGENUMBER> ::= <UNSIGNED INTEGER>
 If no pagenumber is given, 1 is assumed.

6.6 \$ STOP NOT <LINE>
 6.7 \$ STOP LIST

delete breakpoint at line <LINE>
 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>

6.13 asynchronous stop

appears on the terminal
 If the program has been compiled with the debug option, it's execution can be interrupted by two successive control C:

```
!C!C
.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.

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.

THE UNIVERSITY OF MANITOBA

WINNIPEG, CANADA R3T 2N2

15th July, 1975.

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

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.

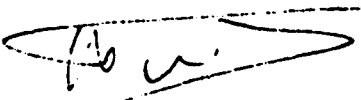
I am very interested in descriptions of other PASCAL Compilers and interpreters for IEM machines. If you have any information on core requirements, compile speeds, and whether or not the full language is supported, for the Grenoble, Stanford or Cambridge projects, I would appreciate hearing from you.

Yours sincerely,

W. Bruce Foulkes

(Enc.)

12

11 



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

