PASCAL NEWSLETTER

May, 1974 Number 2

FROM THE EDITOR

The second newsletter marks the release of PASCAL 2 for CDC CYBER 70 and 6000 series computers under the KRONOS or SCOPE operating systems. Interested CDC users may place orders as explained in the section PASCAL 6000 - 3.4. Also, implementations of PASCAL for other machines have become known through recent correspondence. Further information about these implementations can be obtained by writing directly to the contact given with the description of each implementation.

Please note the following important points.

1) Dr. Wirth, the author of PASCAL, is negotiating with a publisher to print a paperback edition of, "A User Manual for PASCAL" by Jensen and Wirth. People who have received a preliminary version of this manual should not make any further copies of it.

2) The University of Colorado has offered in the past a $10 discount on orders of PASCAL from previous recipients of the package. The discount has been dropped since the new release of PASCAL is more than merely a correction to prior versions. The extra money will be used to defray the cost of this newsletter.

3) A short history of the development of PASCAL is given so that references can be made to the origin of PASCAL compilers on non-CDC computers.

4) A limited number of copies of the first edition of the newsletter are available on request from the editor.

Items of interest or requests for material can be mailed to the editor:

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

or phone: (303) 443-2211, ext. 6934.

HISTORY OF PASCAL

PASCAL is an ALGOL-like programming language with data structure facilities written by Dr. Niklaus Wirth at the Eidgenössische Technische Hochschule (ETH) in Zürich, Switzerland. The original language definition was made in November, 1970, in "The Programming Language PASCAL" published by ETH and later in Acta Informatica 1, 35-63 (1971). The last compiler of this version of PASCAL was released in August, 1972.

In November, 1972, experience gained with the original language revealed certain details of the language that should be changed. This

was done with the publishing of, "The Programming Language Pascal (Revised Report)" in November, "An Axiomatic Definition of the Programming Language Pascal" in December, and the release of an updated compiler dated December, 1972. This compiler implemented all the specifications of the Revised Report except for class variables which conformed to the definition of the original report.

Later, a preliminary version of the PASCAL-P compiler was developed and released to a limited number of sites. Most of the PASCAL compilers implemented for non-CDC machines are based on this compiler and are identical in the form of PASCAL compiled by the December, 1972 release for CDC machines.

May, 1974, brings the release of a completely new PASCAL compiler called PASCAL 2 for CDC machines. Details of the changes made and a description of the materials available are given in the section, PASCAL 6000 - 3.4.

Finally, the PASCAL-P compiler is being rewritten to bring it in line with standard PASCAL. This portable compiler is expected to be available in July.

PASCAL FOR NON-CDC MACHINES

Several sites have implemented PASCAL compilers for computers other than CDC 6000 series machines. The machines represented are the CII IRIS 80, CII 10070, DEC System 10, IBM 360/370, UNIVAC 1108, and XDS SIGMA 7. For further information on these projects, write the contacts given below.

The CII IRIS 80, CII 10070, and XDS SIGMA 7 share the same machine language. Mr. Didier Thibault and Mr. P. Mancel have taken the December, 1972 PASCAL compiler for the CDC machine and bootstrapped it for the CII IRIS 80. This compiler is currently being tested under control of a monitor written for the SIRIS 7 - SIRIS 8 operating system. It generates relocatable binary object code which can be linked by the general linkage editor. It uses the character set ordering as defined by PASCAL on the CDC computer. It allows procedures to be separately compiled and merged at linkage time. It allows all file management compatible with the SIRIS 7 or SIRIS 8 operating system using the S.G.F. assisted file management system distributed by the CII company. It accepts all features of the PASCAL language except the non-dynamic allocation of files.

The compiler consists of 4500 lines of PASCAL code running under control of a monitor (assembly code). The PASCAL program consists of 23,000 machine instructions, and the monitor 1000 machine instructions. It requires 40,000 thirty-two bit words to compile itself. To make this compiler available on other operating systems, the monitor has to be rewritten. This transposition would be easier if a file management system is available on the target machine.

The bootstrap of this compiler was done using the CII IRIS 80 and a CDC machine in parallel. A simulator was not used. It took two experienced programmers 14 man months to complete.

This compiler is currently being tested prior to its distribution. People interested in receiving documentation can place their names on a mailing list by writing:

S.F.E.R./PASCAL
IRIA 15-02
B.P. 5 78150 Le Chesnay
France

Address other correspondence to:

Mr. D. Thibault
17 rue Mayet
75006 Paris
France

The DEC System 10 implementation of PASCAL was developed at the University of Hamburg, Germany by Professor H. -H. Nagel. Work began in April, 1973 with receipt of the preliminary PASCAL-P compiler from ETH. By November, this version would compile itself. As of April, 1974, everything mentioned in the revised PASCAL report of July, 1973, including I/O formats, is implemented with the exception of procedures and functions as formal arguments, arithmetic procedures (SIN, COS, EXP, LN, ARCTAN, ROUND), and GO TO leaving a procedure body (the GO TO EXIT). Work on these areas is in progress.

In particular, the following goals have been reached. The compiler generates in one pass immediately executable (no loader run) reentrant code, generating a sharable pure part and a separate LOW-file containing data. A new feature, INITPROCEDURE, has been implemented to initialize global variables. I/O is possible to standard as well as to declared files. Files may be declared only as global variables. A standard file named TTY is introduced to allow communication with the user terminal. An optional file name may be given in READ/WRITE to use the formatting capabilities of these procedures for all files of CHAR. The actual file name may be indicated at execution time by an optional argument to RESET or REWRITE. The printable upper case ASCII character set is used as internal representation of characters. The appropriate procedures and attributes like READLN(f), WRITELN(f), EOLN(f), etc. have all been implemented. DEC line numbers are recognized, stored, and accessible with a special new procedure GETLINENR. Indexed access to PACKED ARRAY has been implemented. Constant indices are evaluated at compile time. To obtain a completely self-sufficient compiler, the reentrant runtime support is copied out of the compiler into the user's object code file.

A preliminary version of this compiler has already been sent to several sites in the United States (Professor Terry Beyer at the Universite of Oregon, Dr. Donald I. Good at the University of Southern California, and Dr. Frederick A. Hosch at Louisiana State University). A bug contest has revealed several critical and about twelve minor errors in this version which have been corrected in the meantime. The compiler is currently being used in teaching undergraduate students and in small research projects. For further information write:

Professor H. -H. Nagel
Universität Hamburg
Institut für Informatik
2 Hamburg 13, Schluterstrasse 66-72
Germany

Two sites are working on PASCAL implementations for IBM 360 and 370 series machines. Mr. Robert S. Deverill and Mr. Alfred C. Hartmann at the California Institute of Technology have a running version of the preliminary PASCAL-P compiler which will be ready soon. Also, Mr. J. M. Wells and Mr. W. Bruce Foulkes at the University of Manitoba are working on a PASCAL compiler for IBM machines which should be ready soon.

Caltech's version of PASCAL is implemented on an IBM 170/158 running under the OS/VS2 operating system. The environment will operate under any version of the OS operating system. About 270 kilobytes of memory are required to compile the compiler. The compiler uses the recursive descent parsing technique to compile PASCAL programs in a single pass. Only two files, a standard input and output file, are implemented in this version. File declarations are unimplemented, as are exit labels, formal parameter procedures and functions, and the standard functions "succ" and "pred."

For further information write:

Mr. Alfred C. Hartmann
California Institute of Technology
Information Science 286-60
Pasadena, California 91109

The PASCAL compiler at the University of Manitoba had it genesis in 1971 at ETH in Zurich, Switzerland. An early paper, "A Pascal Compiler for the IBM 360/370 Computers," was presented last fall at the Third Manitoba Conference on Numerical Mathematics; reprints should be available by now. The first object programs should be running soon and detailed documentation will be available later this year. For further information write:

Professor J. M. Wells
University of Manitoba
Department of Computer Science
Winnipeg, Canada R3T 2N2

The Univac 1108 implementation of PASCAL was done at the Technical University of Norway by Professor Tore Amble and two of his degree students, Mr. Terje Molster and Mr. Vernar Sundvor. The compiler is based on the preliminary version of the PASCAL-P compiler. All code generated is in the form of subroutine calls, so efficiency of compiled programs is not very high. Packed records, packed arrays, files (except for standard files), formal procedures, and formal functions were not implemented. For further information write:

Professor Tore Amble
Computing Centre
Technical University of Norway
Department of SIN TEF
N-7034 Trondheim-NTH
Norway

PASCAL 6000-3.4

by N. Wirth

An entirely new compiler for the CDC 6000 series of computers has been under development at ETH Zurich for the last 18 months. As predicted last fall and announced in the first issue of the NEWSLETTER, it is released in May 1974. An important development is the definition of a Standard PASCAL: in the interest of portability of programs, we wish to make a clear distinction between Pascal and Pascal-like languages, as several of these have already been proposed. The new compiler adheres to this Standard, and includes some additional facilities clearly labelled as extensions (3.5--3.6). This Standard also includes the definition of a program representation in terms of the ASCII character set.

The new compiler is designed for use under the CDC SCOPE 3.4 operating system with its 64-character set. The decision to adapt PASCAL to the ASCII set and to character sets without explicit control characters has made necessary some changes in the definition of the language. Of particular importance is the decision to eliminate the eol character. It was felt that this change is in the interest of making Pascal less dependent on particular character sets and actual representations of textfiles.

A summary of the changes and innovations of Pascal 6000-3.4 compared to Pascal 6000-3.2 is presented below in an informal, descriptive style. It is divided into the following parts:

1. Notation (representation, character sets)
2. Differences between Pascal 6000-3.4 and Pascal 6000-3.2
3. New facilities of Pascal 6000-3.4
4. New predefined procedures and functions
5. Control statements (SCOPE 3.4)

The new compiler generates relocatable binary code that can be loaded by the standard loader of the operating system. Before execution, the generated code must be linked by the loader with a set of subroutines for input/output handling. Each program operates on files that are declared as formal parameters in its heading, and are substituted with actual files that can be specified in the EXECUTE statement of the control statement record.

Besides some new features, the major advantage of the new compiler is its improved code which makes compiled programs more efficient and more compact. The price for the expanded capabilities is a larger size of the compiler: for average programs, a field length of 60000 (octal) is needed.

The Pascal 6000-3.4 compiler can be ordered from

Ms. Kathleen Jensen
Institut fur Informatik
Clausiusstr. 55
8006 Zurich
Switzerland

The charge for a minitape, tape handling, postage, and documentation is SFr. 100; if a tape is supplied by the requestor, the charge is SFr. 75. (Please send minitapes only!)

In the USA and Canada, orders must be directed to

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

The charge for a minitape, tape handling, postage (North American continent), and documentation is $30; if a tape is supplied by the requestor, the charge is $20.

The system is available in two versions, namely for use with the ASCII character set (CDC-defined collating sequence) or with the CDC scientific character set. When ordering, please specify

PASCAL 6000-3.4 ASCII or
PASCAL 6000-3.4 CDC

Along with the system, the following documentation is provided:

1. A User-Manual. (copying probibited, as we are currently in contact with a publisher who might possibly be able to provide this manual along with the Revised Report in a moderately priced paperback edition.)

2. A description of the contents of the tape with instructions on how to install the Pascal system.

Note: the system also runs under SCOPE 3.2, 3.3, and 3.4 with the 63-character set, with the only restriction being that the % character cannot be used.

1. Notation

1.1 Set union is denoted by + and set intersection by * (instead of $\vee$ and $\wedge$).

1.2 The symbols in the left-hand column may be substituted for those in the right-hand column.

until now	new
¬	not
∧	and
∨	or
≠	<>
≤	<=
≥	>=
{ and }	(* and *)

Note: ∧ and ∨ denote Boolean operations and cannot be used for set operations.

The above table defines a unique, context-independent correspondence between those Pascal symbols which are not available in the international standard of ISO (ASCII) and the ASCII character set. Hence there is a standard representation for a Pascal program in the ASCII character set.

2. Differences between PASCAL 6000-3.4 and PASCAL 6000-3.2

2.1 End of lines in textfiles

The control character eol, which marked the end of a line, has been eliminated. Instead, the following textfile operators are able to recognize and generate line endings:

eoln(f) a predicate function, evaluated while reading a textfile, which indicates whether the end of the current line in the textfile f has been reached. Suppose the buffer variable f↑ is positioned at the character x and that the procedure "get(f)" (or "read") is called in order to access the next character. If x had been the last character in the line, then f↑ = ' ' (blank), and the value of eoln(f) = true. The next call of get(f) (or read) accesses the first character of the next line, and eoln(f) = false (provided the next line is not empty).

writeln(f) a standard procedure that terminates the current line when writing the textfile f.

readln(f) a standard procedrue that skips to the beginning of the next line; the buffer variable f↑ is equal to the first character of the new line.

The usual program schema for sequential reading of a textfile f follows: (x is a variable of type char; P denotes the processing of the (next) character.)

```
reset(f);
while ¬eof(f) do
begin  beginline;
   while ¬eoln(f) do
   begin
      read(f,x);  {read from the textfile f and
                   assign to x; see section 3.1}
      P(x)
   end;
   endline;  readln(f)
end
```

A line ending is represented by a blank. Notice that the following schema can be used when it is not necessary to recognize line endings--i.e. when no special action is required upon encountering an end of a line:

```
reset(f);
while ¬eof(f) do
   begin  read(f,x);  P(x)
   end
```

The following abbreviations may be used:

abbreviated form	expanded form
writeln(f,x1, ... ,xn)	begin write(f,x1,...,xn); writeln(f) end
readln(f,x1, ...,xn)	begin read(f,x1,...,xn); readln(f) end

Note: The first parameter names the relevant textfile (see section 3.1); when it is not of type text, then the file "input" is assumed by reading and the file "output" by writing. Hence,

writeln stands for writeln(output)

and

readln stands for readln(input)

2.2 The program heading

PASCAL 6000-3.4 requires the specification of a program heading. The form is:

```
program p(x1,x2, ... ,xn);
```

where p is the name of the program and x1...xn are formal file parameters (n≥1). x1...xn are available to the program, but also exist outside of the program; hence, they are called external files (as opposed to local files).

be used in the control statement: (see section ")

EXECUTE,p(f1, ... ,fn)

where f1...fn are file names, i.e. the actual parameters corresponding to the formal parameters x1...xn.

The following rules hold:

a) The program heading must contain the formal parameter "output".

b) As with any other variable, the files denoted by the names x1...xn must be declared as file variables in the main program. The exception occurs with the files "input" and "output" which are automatically predeclared as:

```
var input,output: text;
```

c) If any actual parameter fi in the EXECUTE statement is left empty, the corresponding formal parameter xi in the program heading is then assumed as the actual "logical file name".

d) Only one "logical record" will be read from the actual file INPUT--i.e., the next EOR mark appears as EOF in a Pascal program.

e) If a file xi is to be opened for reading only, then this must be indicated by an asterisk following the file parameter in the program heading. (This is necessary, if actual files are Permanent Files with Read Permission only.)

Note: Rules a)--e) are specificly for the CDC implementation. A consequence of rule c) is that a program with a program heading:

```
program standard(input,output);
```

can be called simply with the control statement:

EXECUTE,STANDARD.

or even

EXECUTE.

when the standard SCOPE files INPUT and OUTPUT are intended. Note that the file specifications [IN] and [OUT] of PASCAL 6000-3.2 are eliminated.

2.3 The label declaration part

Every label must be declared. Consequently, the symbol exit is eliminated from the goto statements. If a label L (an

unsigned integer) marks a statement in the statement part of a block A, then L must be declared in the label declaration part of A.

Goto statements should be avoided whenever possible, thereby making the computational structure of the program more transparent. Jumps from outside of a structured statement into that statement are not allowed.

2.4 The standard type "alfa"

Due to the introduction of packed arrays (see section 3.2), the standard type alfa can be explicitly declared as

```
type alfa = packed array[1..10] of char;
```

Alfa values must, therefore, comply with the rules for packed arrays. In particular, assignments can be made only between identical types. That is, an assignment to an alfa variable a

```
a := <character string>
```

is only allowed when the character string has exactly 10 characters.

Likewise, alfa may no longer be regarded as a scalar type. Consequently, a result type of a function cannot be of type alfa, nor can the argument of the function ord.

2.5 Pointer types and class variables

The concept of a class variable is eliminated. Instead of

```
type pointer = ↑classvariable;
var  classvariable: class of T;
```

is now simply

```
type pointer = ↑T;
```

Apart from this, the facilities for pointer handling remain the same. The modification affects only the declaration part.

2.6 The value part

PASCAL 6000-3.4 has no value part as did PASCAL 6000-3.2. (A more general substitute facility is under consideration.)

3. New facilities of PASCAL 6000-3.4

3.1 Read and Write

The standard procedures read and write can apply to any textfile, not just to the files "input" and "output". The first parameter names the textfile; when it is not a file variable, then the file "input" is assumed by read and the file "output" by write. For example,

```
write(x,y)      stands for      write(output,x,y)
```

(Also see section 2.1 for readln and writeln.)

3.2 Packed arrays

The symbol packed before the symbol array means that the storage requirements for the array structure should be minimized. It has no other influence on the meaning of the program. One should keep in mind that accessing an element of a packed array can take more time than accessing one from an unpacked array. However, the gain in storage can be very great (up to a factor of 60 in the case of a Boolean array). For example, the variable x

```
x: packed array [1..n] of 0..999
```

requires 6 times less storage than the same variable would were it not packed (or if the component type had been specified as "integer"). The reason is that a number between 0 and 999 can be expressed with 10 binary digits; hence, 6 such numbers can be stored in one 60-bit word.

The packed array is especially important in connection with character strings, where each group of 10 6-bit characters are packed into one word. The standard type "alfa" is a special case of a string (see 2.4): an alfa value fits exactly into one word.

A restriction common to all packed structures is that no component of such a structure (e.g. an element of a packed array) may appear as an actual parameter when the corresponding formal parameter is specified as a var parameter (variable parameter).

3.3 Record types with a variant part but without a tag field

Obligatory in PASCAL 6000-3.2 is the presence of a tag field when a record type has a variant part. For example, the tag field x was necessary in the following declaration.

```
R: record a: T1;
       case x: sex of
         male: (bm: T2);
         female: (bf: T3)
   end
```

In PASCAL 6000-3.4 the tag field x may be omitted, thereby simplifying the above case-clause to:

```
case sex of
```

(where "sex" is a programmer-defined type identifier). The advantage is that R then requires less storage; the disadvantage is that it is impossible to establish from the value of R alone which variant is present. (e.g. one can no longer ask: "if R.x=male then ...") Therefore, one should use this new flexibility only with the greatest of care.

3.4 Values of type set

Given is a scalar type W with the values w1,w2,...,wn. The set m = [wi,w(i+1), ... ,w(j-1),wj] can be more simply expressed with the notation:

```
m = [wi..wj]
```

where wi and wj are arbitrary expressions of type W, and m is a variable of type set of W.

3.5 External procedures and functions

In PASCAL 6000-3.4 it is possible to call external, separately compiled procedures and functions. One needs only to introduce the name of the procedure (function) by a pseudo-declaration in the program heading. This enables Pascal programmers to build and access program libraries. The user is however cautioned to use great care, for the compiler no longer has the opportunity to check the correspondence between actual and formal parameters. A seperate write-up documenting this facility is in preparation. Note that external procedures and functions are not a facility of Standard Pascal.

3.6 Segmented files

The CDC operating system allows a file to be subdivided into segments of varying lengths, where each segment is a "logical record" in CDC SCOPE terminology. In PASCAL 6000-3.4 these divisions are transparent when the file is declared as segmented. Example:

```
var f: segmented file of T;
```

A number of operations are available to end a segment when generating a file, and to recognize segments and their boundaries when reading a file--regardless of the component type.

putseg(f) completes the generation of the current segment when writing the file f.

eos(f) is a Boolean function indicating whether the end

of a segment has been reached while reading the file f. Assume that the buffer variable f↑ = x (of type T), and that "get(f)" is called. If x was not the last element of the segment, f↑ is the value of the next element. If x was the last element of the segment, eos(f) is "true" and the value of f↑ is undefined.

getseg(f) initiates the reading of the next segment of the file f. f↑ is the first element of the (new) segment. If there is no next segment, "eof(f)" is "true".

The following program schema illustrates the sequential reading of a segmented file f:

```
reset(f);
while ¬eof(f) do
begin  beginsegment;
   while ¬eos(f) do
   begin
      R(f↑);  get(f)
   end;
   endsegment;  getseg(f)
end
```

An advantage with segmented files is the possibliity of positioning the reading and writing head (relatively) quickly to any segemnt in the file. For the purposes of reading and (re)writing a segmented file, the standard procedures getseg and rewrite are extended to accept two arguments.

getseg(f,n) initiates the reading of the nth segment counting from the current position of the file. n>0 implies counting segments in the forward direction; n<0 means counting them backwards; and n=0 indicates the current segment. Note: getseg(f,1) is equivalent to getseg(f).

rewrite(f,n) initiates the (re)writing of f at the beginning of the nth segment counting from the current position. Note: rewrite(f,1) is not equivalent to rewrite(f). The latter causes initiation of (re)writing at the very beginning of the entire file.

Since files are organized for sequential (forward) processing, one should not expect getseg and rewrite to be as efficient for n<=0 as they are for n>0.

Observe the following rules:

1) eof(f) always implies eos(f).
2) get(f) is applicable only when eos(f) = false.
3) put(f) is applicable only when eos(f) = true.

4) getseg(f) is applicable only when eof(f) = false.
5) The procedures putseg(f), getseg(f), rewrite(f,n) and the function eos(f) can only be applied to segmented files.
6) Segmented files are not part of Standard Pascal; they are an extension to the language and oriented towards the CDC operating system.

4. New Predefined Procedures and Functions

4.1 Procedures

readln(x1, ... ,xn) — see 2.1
writeln(x1, ... ,xn)

getseg(f), getseg(f,n)
putseg(f) — see 3.6
rewrite(f,n)

linelimit(f,n) causes the program to terminate when the textfile f has more than n lines. Note: the call "linelimit(output,1000)" is automatically executed at the beginning of a program.

page(f) causes a jump to a new page when the textfile f is being printed.

halt terminates the execution of the program and issues a post-mortem dump.

message(s) writes the string s into the dayfile.

time(a) assignes to the alfa variable a the current time in the form: ' hh.mm.ss.'

date(a) assignes to the alfa variable a the current date.

dispose(p) informs the memory management that the variable referenced through the pointer p will no longer be needed. "dispose(p)" is in a certain sense the inverse of "new(p)".

4.2 Functions

eoln(f) — see 2.1
eos(f) — see 3.6

card(S) equals the cardinality of the set S (i.e. the number of elements contained in the set S).

undefined(x) is a Boolean function. Its value is true when the value x (of type real) is "infinite" or "indefinite" (see CDC Manual). These values arise in the cases of overflow and division

by zero.

expo(x) — is an integer function yielding the exponent of the floating-point representation of the real argument x.

clock — is a function without any parameters. It gives the current processing time in milliseconds.

Note: getseq, putseq, linelimit, halt, message, time, date, eos, card, undefined, expo, and clock are not Standard Pascal, but represent additional features of the CDC implementation. Hence, they must be avoided in programs that are supposed to be portable.

5. Control Statements

In installations which keep the PASCAL system stored as a Permanent File, the compiler is used through the following control statements.

```
ATTACH,COMP, ...
COMP.
ATTACH,LIB, ...
LOAD,LGO,LIB.
EXECUTE.
```

The additional information needed to complete the ATTACH statements must be supplied by the individual computer installations which make PASCAL available. The instruction COMP calls the compiler and can be provided with parameters as follows:

```
COMP(x,y,z)

x = source program      (default = INPUT)
y = listing             (default = OUTPUT)
z = binary code         (default = LGO)
```

The instruction EXECUTE initiates the execution of the PASCAL program, which may call any subroutine in LIB. It has the general form:

```
EXECUTE,p(f1, ... ,fn)
```

where p is either empty or the name of the program given in the program heading. f1...fn are the actual external files (see 2.2).

A Standard Pascal job has the following parts:

```
control statements
end of record        (7/8/9)
PASCAL program
end of record        (7/8/9)
data                 ( = file INPUT)
end of file          (6/7/8/9)
```

Due to the interest expressed by several people to make Pascal available on other computers, the Pascal-P system was developed by U. Ammann as a side-product of the 6000-3.4 compiler project. Its purpose was to provide with minimal effort the means to transport Pascal to other systems with also minimal effort on the part of the receiver. This resulted in the Pascal-P compiler which generates code for a hypothetical stack computer M. This computer is described as a short Pascal program that represents an interpreter. It is combined with a loader that loads the generated code, which is a string of printing characters (i.e. a text). The only effort on the part of a receiver of this system is then to code the loader/interpreter in an efficient manner on his available computer. Of course, the system obtained in this way is interpretive and inherently slow; however, for short programs its speed proved to be quite acceptable.

Currently, the Pascal-P compiler is being modified in order to
1. comply with Standard PASCAL,
2. overcome some shortcomings of the present version,
3. be expressed entirely in terms of ASCII characters.

The new Pascal-P compiler is planned to be available in July 74, together with documentation. It processes (standard) Pascal with a few restrictions (e.g. no packed structures, no formal procedures and functions, no file declarations).

The Pascal-P approach is quite adequate and convenient, if efficiency of program execution is of no great significance. However, if the development of a high-quality compiler is the objective, a bootstrapping process on the basis of an interpretive Pascal-P system is very costly, and involves a large amount of reprogramming. It is clear that a different approach to the transportation of compilers themselves should be investigated.

A project has now been started with the aim to construct a machine-independent Pascal compiler which can be extended (completed) into a high-quality compiler for almost any real computer with a reasonable amount of additional effort. The system is developed by H.H. Nageli at ETH Zurich, with cooperation from Cambridge University (England), where the product will be subjected to its first test, namely its adaptation to the IBM System/360.

ADDRESS CORRECTIONS

Please help us keep our mailing list up to date by mailing this page back if your address is incorrect or you are no longer interested in PASCAL. Additional names of interested parties are welcome also.

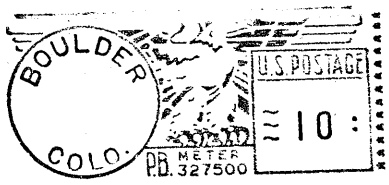
Drop my name.

Correct address as indicated.

Add the following names.

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

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



PASCAL NEWSLETTER