PASCAL NEWSLETTER

January 1974 Number 1

FROM THE EDITOR

This is the first issue of a newsletter sent to users and other interested parties about the programming language PASCAL. Its purpose is to keep the PASCAL community informed about the efforts of individuals to implement PASCAL on different computers and to report extensions made to the language. It will be published at infrequent intervals due to the limited manpower.

Everyone is encouraged to report to the editor any item of interest to the community. There are many examples given below. Of particular interest are reports of successful implementations on other computers that the author is willing to distribute and modifications to the CDC version to surmount problems or add features. Also, programs written in PASCAL are of interest. However, a plethora of incompatible dialects of PASCAL must be avoided.

The University of Colorado, thru the editor, is willing to support the distribution of PASCAL programs and documents subject to the limitations of its hardware. This is a CDC site with seven track industry compatible NRZI tape drives. To help defray our costs, there is a charge of 5¢ per page for documents, $5 for copying and postage for a magnetic tape from a user and $15 if the University supplies the tape (a mini reel). Send to:

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

CURRENT PASCAL COMPILER

The current compiler is dated December 15, 1972, and now incorporates a correction written November 15, 1973. It is available on a 556 bpi SCOPE 3.2 internal binary tape. All CDC 6000 sites should be able to read the tape under any standard CDC operating system. Other tape formats can be arranged on request.

There are six files on the tape. The first two are the source and binary of PASCAL as distributed by Dr. Wirth, of Switzerland, the author of the language. The next two files are the University of Colorado version of the source and binary of PASCAL. The changes are confined to: 1) adapting PASCAL to KRONOS 2.0, 2) deleting the frequency count feature, and 3) providing alphabetic alternatives to some special characters for terminal users. The source file is an UPDATE version 1.2 sequential OLDPL of the 72/12/15 release with the 73/11/15 correction applied as a modification.

All PASCAL users are urged to use this as a base from which to modify the compiler. The fifth file is a PASCAL Manual adapted from the one written by Wilhelm Burger at the University of Texas at Austin. The last file is the text of the modsets incorporated in the OLDPL.

The following items are supplied in the documentation package.

TITLE	DATE	PAGES
The PASCAL System, Documentation, and Literature	Nov 73	4
The PASCAL Distribution Tape	Dec 73	1
Notes accompanying the PASCAL Tape of	15 Dec 73	8
A note to users of the PASCAL Language	15 Dec 72	6
Planned changes to the programming language PASCAL	Jun 72	7
Changes to the programming language PASCAL	15 Feb 72	8
The PASCAL Operating System POSYS	28 Oct 71	20
Advantages of the value parameter over the constant parameter	5 Jul 72	3
Run-Time Check Options	Dec 72	1
How to use the PASCAL 6000 System	23 Jun 72	3
The Standard Procedure WRITE and Post-Mortem Dump	23 Jun 72	4
The Programming Language PASCAL (Revised Report)	Nov 72	53
An Axiomatic Definition of the Programming Language PASCAL	Nov 72	32

This version of the compiler does not implement class variables as described in the Revised Report. The following description of them comes from the original PASCAL report.

6.2.5 Class types

A class type definition specifies a structure consisting of a class of components, all of the same type. The number of components is variable; the initial number upon declaration of a variable of class type is zero. Components are created (allocated) during execution of the program through the standard procedure new. The maximum number of components which can thus be created, however, is specified in the type of definition.

<class type> ::=class <maxnum> of <type>

<maxnum> ::= <integer>

6.2.6 Pointer types

A pointer type is associated with every variable of class type. Its values are the potential pointers to the components of that class variable (cf.7.5) and the pointer constant nil designating no component. A pointer type is said to be bound to its class variable.

<pointer type> ::= ↑<class variable>

<class variable> ::= <variable>

7.2.4 Referenced components

Components of class variable are referenced by pointers.

<referenced component> ::= <pointer variable>↑

<pointer variable> ::= <variable>

Thus, if p1 is a pointer variable which is bound to be class variable v, p1 denotes that variable and its pointer value, whereas p1↑ denotes the component of v referenced by p1.

Examples:

p1↑. father

p1↑. elder sibling↑. youngest child

10.1.2 Class component allocation procedure

new(p) allocates a new component in the class to which the pointer variable p is bound, and assigns the pointer designating the new component to p. If the component type is a record type with variants, the form

new(p,t) can be used to allocate a component of the variant whose tag field value is t. However, this allocation does not imply an assignment to the tag field. If the class is already completely allocated, the value nil will be assigned to p.

An error in code generation was recently found and corrected as shown below. It occured when an integer variable I and a real variable X are used in expressions like I+1+X.

Procedure SIMPLEEXP before fix:

```
PROCEDURE SIMPLEEXP ;
VAR LATTR :  ATTR ; LADOPCL  :  SHRTINT ; LFB,BT1,BT2 : BOOLEAN ;
BEGIN LFG := FALSE ;
  IF NO = 7 THEN {ADDOP}
  BEGIN IF CL = 2 THEN LFG := TRUE ELSE IF CL = 3 THEN ERROR(51) ;
    INSYMBOL ;
```

```
  END ;
  TERM ;
  IF LFG∨(NO = 7) THEN
  BEGIN WITH LATTR DO
    BEGIN TYPTR := GATTR.TYPTR ; KIND :=LVAL ; CTERM :=0 ;
      IF TYPTR ≠ NIL THEN
      BEGIN TRANSFER(GATTR,RP) ;
        IF LFG THEN
        BEGIN GEN15(13B,0,0,0) ;
          IF (TYPTR↑.FORM = NUMERIC)∨(TYPTR = REALPTR) THEN
          GEN15(37B,RP,0,RP) ELSE ERROR(50)
        END
      END

  END ;
  WHILE NO = 7 DO
  BEGIN LADOPCL := CL ; INSYMBOL ; TERM ;
    IF (LATTR.TYPTR ≠ NIL)∧(GATTR.TYPTR ≠ NIL) THEN
```

Procedure SIMPLEEXP after fix:

```
PROCEDURE SIMPLEEXP ;
VAR LATTR : ATTR ; LADOPCL : SHRTINT ; LFG,BT1,BT2 : BOOLEAN ;
BEGIN LFG := FALSE ;
  IF NO = 7 THEN {ADDOP}
  BEGIN IF CL = 2 THEN LFG := TRUE ELSE IF CL = 3 THEN ERROR(51) ;
    INSYMBOL ;
  END ;
  TERM ;
  IF LFG∨(NO = 7) THEN
  BEGIN WITH LATTR DO
    BEGIN TYPTR := GATTR.TYPTR ; KIND := LVAL ; CTERM :=0 ;
      IF TYPTR ≠ NIL THEN
      BEGIN TRANSFER(GATTR,RP) ;
        IF LFG THEN
        BEGIN GEN15(13B0,0,0) ;
          IF (TYPTR↑.FORM = NUMERIC)∨(TYPTR = REALPTR) THEN
          GEN15(37B,RP,0,RP) ELSE ERROR(50)
        END
      END
    END ;
    WHILE NO = 7 DO
    BEGIN LADOPCL := CL ;
      IF LATTR.KIND = LVAL THEN
         IF LATTR.CTERM ≠ 0 THEN
            BEGIN GEN30(71B,0,0,LATTR.CTERM) ; GEN15(36B,RP,RP,0);
               LATTR.CETERM := 0
            END;
        INSYMBOL; TERM;
      IF (LATTR.TYPTR ≠ NIL)∧(GATTR.TYPTR ≠ NIL) THEN
```

COST OF THE PASCAL PACKAGE

There is a charge of $30 the first time a PASCAL tape and documentation package is sent. Deduct $10 if a tape is supplied and deduct $10 if you have previously received PASCAL from us.

FORTHCOMING VERSIONS OF THE COMPILER

An entirely new compiler called PASCAL 2 is under development. It implements the language PASCAL as defined in the Revised Report (Nr. 5) with a few extensions (see below). It is implemented for the operating system SCOPE 3.4 and will be available in two versions:

1) For the CDC scientific 64-character set, and
2) For the ASCII 64-character set (with CDC's ordering).

This implies that no explicit line control characters (eol) are available; instead, ends of lines in textfiles are generated and recognized by additional standard procedures and functions. This, unfortunately, requires changes in most existing programs - although they may be clerical only. Hence, the distinct name PASCAL 2 was chosen.

The (other) principal new characteristics of the compiler are:

1) It generates relocatable binary object code which can be loaded by the standard CDC loader.

2) It allows procedures (and functions) to be separately compiled and merged at load-time.

3) It provides a facility to use subroutines written in other languages. In these cases, the standard FTN calling sequence will be generated.

4) It introduces packed arrays. They can be treated like regular arrays, but will be allocated with as many components as possible packed into each word. Accordingly, access to individual elements will be slower.

5) It introduces so-called segmented files; each segment corresponds to a logical record (in CDC terminology).

6) External files may be passed to the program as parameters in a program heading.

We hope to be able to release the new compiler by May, 1974 along with a User Manual.

The PASCAL-P system is a compiler which generates code (so-called P-code) for a simple, hypothetical stack computer. This computer is described in the form of a PASCAL program as a loader and interpreter of P-code.

The PASCAL-P system was developed in order to simplify the implementation of PASCAL on other machines. The method to proceed is in general (i.e., without access to a CDC 6000 computer) the following:

1) Program the loader interpreter in any (probably assembly) language for the target machine M . PASCAL-programs can now be executed interpretively on M , since the PASCAL-P compiler is available in P-code.

2) Rewrite the PASCAL-P compiler by replacing the P-code generators by routines generating code for machine M.

3) Interpretive compilation of the modified compiler then yields a compiler in P-code which generates M-code.

4) Recompilation of the modified compiler by itself then results in a PASCAL compiler in the form of M-code and generating M-code.

The PASCAL-P system is available under the same conditions as the CDC compiler. It is delivered in source form (a PASCAL program) and in P-code form (coded as a string of characters).

Note: We recommend that this system be ordered only by people seriously considering to implement PASCAL on their computer.

OTHER PASCAL COMPILERS

Mr. Wilhelm Burger at the University of Texas at Austin has made extensive modifications to PASCAL and has written a manual that is available in machine readable form. The University of Colorado has adapted this manual so that it corresponds to the version we release. Mr. Burger's PASCAL incorporates the following extensions. It allows external procedures, segmentation of programs into overlays which can be called sequentially, and partial compilation of declarations and procedures to allow them to be used later. Compiler options can be set on the control card. Several new convenient procedures and functions are defined such as a random number generator, clock, memory dump, and tracing of procedure calls.

A tape of Mr. Burger's PASCAL system can be obtained from the University of Colorado or one can write directly to:

Mr. Wilhelm F. Burger
The University of Texas at Austin
Department of Computer Science
Austin, Texas 78712

An IBM 370 version of PASCAL that runs interpretively at speeds comparable to PL/I for student jobs is available. For further information, write directly to:

Mr. Al Hartmann
Mail Code 286-80
California Institute of Technology
Pasadena, California 91109

MODIFICATIONS TO PASCAL

The University of Colorado has developed and distributes with the PASCAL system the following mods which are of interest to all PASCAL users.

FUNCTION	DATE	PAGES
Adapt POSYS to KRONOS 2.0	30 Jan 73	5
Remove frequence count feature	30 Jan 73	1
Print PMD message on OUTPUT in addition to dayfile	6 Feb 73	2
Add alphabetic tokens equivalent to special symbols	30 Jan 73	1

In addition, a 23 page copy can be made of the following mods from Professor Hellmut Golde at the University of Washington in Seattle, dated November, 1973.

Incorporate cross reference program
Abort after too many compilation errors
Delete frequency count feature
Implement 64 character set for SCOPE 3.4
Adapt POSYS to SCOPE 3.4
Require EOR between program and data on all files
Yield error on attempt to read past EOF
Print PASCAL error message on abort
Accept alphanumeric tokens for special symbols
Add line numbers to compiler listings
Scan only 72 columns of source

OTHER DOCUMENTATION

Copies can be made from the following items in our files.

Author and Title	DATE	PAGES
N. Wirth, ETH, "Program Development by Step-wise Refinement"	Jan 71	24
N. Wirth, ETH, "The Design of a PASCAL Compiler," submitted to Software-Practice and Experience	Jul 71	26
N. Wirth, Stanford University, "On PASCAL, Code Generation, and the CDC 6000 Computer"	Feb 72	40
N. Wirth, ETH, "The Programming Language PASCAL"	Aug 72	61
A. Mickel, University of Minnesota, "PASCAL at the University of Minnesota"	Sep 72	16

Author and Title	Date	Pages
N. Wirth, ETH, "The Programming Language PASCAL (Revised Report)"	Nov 72	53
W. Burger, University of Texas at Austin, "PASCAL Manual," a complete manual for their local version	Jul 73	68
H. Golde, University of Washington, "PASCAL-W, Users Manual," a list of changes to the revised report for their local version	Sep 73	24
K. Jensen and N. Wirth, ETH, "A User Manual for PASCAL," a preliminary copy to be released with the next version of the compiler	Oct 73	87

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

PASCAL NEWSLETTER

Notes Accompanying the PASCAL Tape of December 15, 1972

These notes contain a description of the PASCAL tape of December 15, 1972, and instructions how to use it.

Please observe that the notes entitled "Notes Accompanying the PASCAL Tape of August 11, 1972" are obsolete.

December 15, 1972

U. Ammann
Fachgruppe Computer-Wissenschaften
Eidg. Technische Hochschule
Clausiusstrasse 55
CH-8006 Z ü r i c h

Contents of the PASCAL Tape of December 15, 1972:

The tape contains four files.
File no. 1 and file no. 3 are identical, and so are file no. 2 and file no. 4.

File no. 1 contains source programs only and consists of the following 6 records:

Record no.	Contents
1	POSYS
2	PASCAL Compiler
3	five COMPASS library routines
4	FQC (frequency counting PP routine)
5	FQCOUT (outputs program profiles)
6	PMD (post-mortem dump routine)

File no. 2 contains the binary PASCAL system and consists of the following 12 records:

Record no.	Contents
1	POSYS
2	Header of Compiler
3	Compiler
4-8	Library functions
9	Header of FQCOUT
10	FQCOUT
11	Error messages
12	PMD

How to use the tape

The following pseudo control cards suggest a way to catalog, assemble and compile the various programs:

```
JOBCARD: CM47000, TIME = 130 SEC, 1 TAPE UNIT.
RFL(10000)
REQUEST(TAPE,HI)   PASCAL TAPE OF DEC. 15, 1972.
COMMENT.
COMMENT.1) CATALOG TAPE INFORMATION.
COMMENT.----------------------------
ASSIGN PF DEVICE TO THE FOLLOWING FILES:
   POSYS,SRCOMP,PASCLIB,FQC,FQCOUT,PMD,PASCAL.
COPYBR(TAPE,POSYS)
CATALOG(POSYS,PASCAL DEC1972,CY=10)
COPYBR(TAPE,SRCOMP)
CATALOG(SRCOMP,PASCAL DEC1972,CY=20)
COPYBR(TAPE,PASCLIB)
CATALOG(PASCLIB,PASCAL DEC1972,CY=30)
COPYBR(TAPE,FQC)
CATALOG(FQC,PASCALFREQCOUNT DEC1972,CY=10)
COPYBR(TAPE,FQCOUT)
CATALOG(FQCOUT,PASCALFREQCOUNT DEC1972,CY=20)
COPYBF(TAPE,PMD)
CATALOG(PMD,PASCAL DEC1972,CY=50)
COPYSF(TAPE,PASCAL)
CATALOG(PASCAL,PASCALBIN DEC1972,"READ-ONLY")
RETURN(TAPE)
COMMENT.
COMMENT.2) ASSEMBLE, COMPILE.
COMMENT.---------------------
REWIND(POSYS,SRCOMP,PASCLIB,FQC,FQCOUT,PMD)
RFL(47000)
COMPASS(I=POSYS,B=0)
PASCAL(P=SRCOMP,OPT=NOGO)
COMPASS(I=PASCLIB,B=0)
COMPASS(I=FQC,S=SCPTEXT)
PASCAL(P=FQCOUT,OPT=NOGO)
COMPASS(I=PMD,B=0)
```

In order to install the frequency counting facility, execute the following steps:

- Read the next four pages.
- Make the necessary changes to FQC according to the demands of your operating system.
- Add the assembled version of your FQC to your system library.

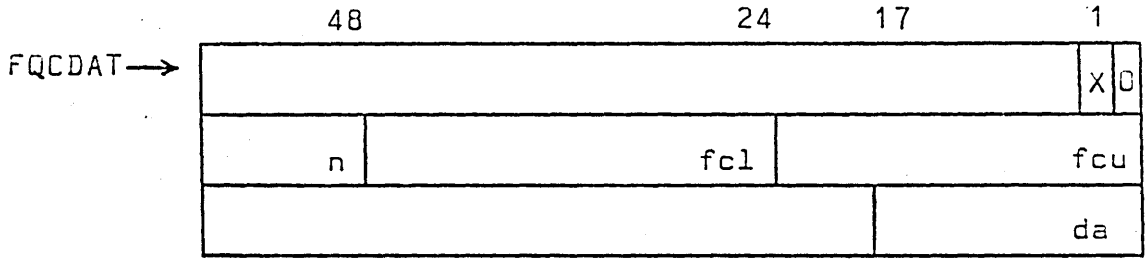
4. Frequency Counting

The PASCAL system in its ETH implementation offers a facility which exposes to the user how much time (in percents of the total time used) each line of his program consumed during execution.

The frequency counting mechanism consists of four routines: FQC, CPFQC1, CPFQC2, and FQCOUT.

FQC is a PP program which procedure GO calls immediately before activating the user program. FQC works with an array of at most 1600_{10} counters. Approximately every 34 μsec FQC increments by 1 the counter indexed by the P register's current value, until it is recalled by GO. Then the contents of the counters are dumped into central memory.

CPFQC1 sets up a table of parameters for FQC. This table is called FQCDAT and has the following format:



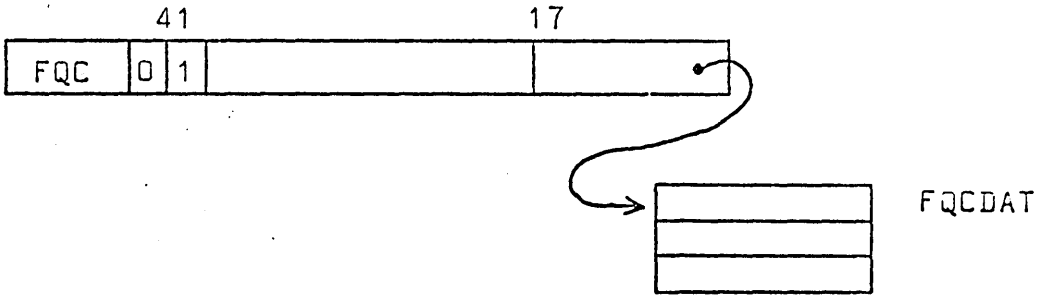
X = 0 for call of FQC, X = 1 for recall of FQC
n = $\log_2$(block length)
fcl = lower bounding address of supervised code area
fcu = upper bounding address of supervised code area
da = dump address = address from which on FQC is to dump the counter values.

CPFQC1 increases the specified FCU parameter (or its default value), yielding fcu, such that the following relation holds:

$(fcu - fcl) = m * 2^n \ \& \ m \leq 1600_{10}$,

where n is the smallest integer that satisfies the relation.

FQC is called by setting [RA+1] to



CPFQC2 is called after the recall of FQC. Its only function is to load and execute and pass parameters to FQCOUT.

FQCOUT, finally, is a kept PASCAL program that outputs the dumped counter values. Its main program variables have been initialized by FQC and CPFQC2 before it is loaded. If OPT=FC* was specified, FQCOUT takes as input the listing of the user program which the compiler wrote on the file FQCFILE, and outputs the profile of the program.

4.1. FQC - The Frequency counting PP Program

(by W. Bächi, Fachgruppe Computer-Wissenschaften, Eidg. Technische Hochschule, Clausiusstr. 55, CH-8006 Zürich)

FQC consists of four parts. The first one is used to prevent more than one copy of FQC being active at any time. The second part checks the parameters obtained from the calling CP program and starts the third part, which contains the 34 μsec main loop. When FQC is recalled, control is given to the fourth part which dumps the counter values to the designated central memory area.

The parameters supplied are used as follows:

$$\text{blocksize} = 2^n$$

length of supervised central memory area,

$$\text{length} = ((\text{fcu} - \text{fcl})//\text{blocksize}) * \text{blocksize}$$

To each CP program block a 24-bit counter in the PP memory is assigned. They are arranged in two arrays for convenient indexing. When FQC in its main loop has read the CPU's P-register the index to the appropriate counter is

$$(P-(\text{fcu} - \text{length}))//\text{blocksize}$$

provided that it is greater than zero and (P-fcu) is negative. Otherwise, the index value -1 is used. Thus the time spent outside the given limits is recorded in the first counter.

Now we go through the four parts of FQC in detail. First, when FQC is entered, a test is made whether FQC is already used by another job. This is done by inspecting byte C.FQCLCK of word T.FQCLCK in CMR using channel CHLOCK as interlock. A non-zero value indicates another FQC already being active. To prevent PP saturation, the request is not honoured in that case, and FQC puts itself into the PP delay queue with a delay of 5 sec., and drops. Otherwise, the interlock flag is set and the second part acquires control.

It starts with fetching the parameters and checking them. If any error occurs, a simple error routine is called which puts into the dayfile the FQC program address where the error was found. Some shift instructions are then set up for division by and multiplication with the blocksize. The counters are initialized to zero, the completion bit is set and CP program execution is requested to resume by the monitor function RCLCP.

Then the main loop starts with testing whether CPU-A or CPU-B is active at the control point by inspection of the CPU status word at T.CPT1 of CMR. Depending on which CPU is active the appropriate RPN instruction is executed to get the P register's value, thus giving the index to the counter which has to be incremented. If neither CPU is active at the control point, several checks have to be made to enquire what was happened. The control point status word is read to see whether the error flag is set, in which case FQC drops. The status word also contains the move flag which is tested. If it is set, a pause request is issued and FQC waits until the move flag is clear or the error flag is set. In the latter case, FQC drops. The parameter address must be updated. Finally, the status word is inspected to see if the CP program has requested to dump the counters (X = 1), so that the fourth part should be executed. If none of the above conditions exist, the main loop is restarted.

Before dumping the counters into central memory, the fourth part checks if the dump area lies within the user's field length. The counters are then expanded to the 60 bit central memory word size and transferred to the array whose address has been given by da. The completion bit is set and FQC drops after having cleared the interlock byte giving another FQC a chance of running.

Fachgruppe Computer-Wissenschaften
ETH, Zürich

N. Wirth
15.12.72

A Note to Users of the PASCAL Language

Efforts to implement PASCAL on various computers have lately been initiated at several places. It is my conviction that the utmost should be done to achieve full compatibility among the different systems. An axiomatic definition of the language together with a set of recommendations for implementation standards has therefore been established and is due to appear shortly:

> C.A.R. Hoare and N. Wirth, "An axiomatic definition of the programming language PASCAL", Berichte der Fachgruppe Computer-Wissenschaften, ETH, No. 6 (Dec. 1972)

The axiomatic definition method as well as past experience with PASCAL showed that certain details of the language should be revised. In view of the growing number of implementations it became highly desirable to define these revisions at this time. The result is the accompanying Revised Report. The main changes are summarised informally below.

Along with the decision to define a revised PASCAL language went the decision to write a completely new compiler, although the small changes themselves would not have required such a procedure. This new compiler will be designed in a functionally structured fashion, and it should be particularly well suited for adaptation by bootstrapping to different computers. This new compiler, however, will not be available for some time, partly because also other changes will be implemented, such as the generation of binary, relocatable object code and the facility of part compilation.

In order to make Revised PASCAL widely available as soon as possible and to facilitate the gradual adaptation of already existing programs, a new version of the existing compiler was prepared (Version 15.12.71). To keep our efforts for this "temporary tool" reasonably small, however, the change concerning class variables (No. 4 below) is not incorporated, and in this respect the compiler still follows the specifications of the original report.

The Revised Report does not contain any specifications about how to access the compiler. It was felt that this information should be provided by individual installations in accordance with their operating systems.

Summary of changes of the language

(cf. also "Preface to the Revised Report")

1. Procedure parameters

"Constant parameters" are replaced by "value parameters" (in the sense of ALGOL 60). A value parameter denotes a local variable to which the value of the corresponding actual parameter is assigned upon initiation of the procedure. This implies that assignments to value parameters are allowed. It should be noted that this concept requires that a local copy of the value of the parameter is made. A change of existing programs is necessary in the following three cases:

- If the formal parameter is specified by the symbol const, then this symbol must be removed.
- If an assignment to a constant parameter is made, then this is against the rules of the language; however, such illegal assignments were not detected by the old compiler in the case of structured parameters (arrays etc.), and as a consequence do occur in existing programs. Such assignments are legal in the new language, but are made to the local variable representing the parameter; this may not necessarily correspond to the intensions of the programmer.
- In ambiguous constructions such as

  ```
  procedure P(var i: integer; r: real)
  ```

  r was considered as a variable parameter, but is taken to be a value parameter in the new language: the symbol var has effect only up to the next semicolon. All parameters not preceded by a specificator are considered to be value parameters.

2. Files

In order that the buffer variable f↑ of a file f and the standard function eof(f) have always a defined value when a file is read, an implicit assignment of the first element of the file f to the buffer variable f↑ is performed (by an implicit call of get(f)) in the following cases:

- at the start of the program for the standard file input ,
- at the start of the program for every file specified by [in] ,
- after every call of the standard procedure reset(f) .

If a file is to be rewound for rewriting, then the calls to reset must be replaced by calls of the new standard procedure rewrite. The further changes necessary in existing programs essentially consist of the removal of the first get(f) (or read(ch)) statement.

Example 1:

```
get(f);
while ¬eof(f) do
   begin S(f↑); get(f)
   end
```

Necessary change: remove first line.

Example 2:

```
read(ch)
while ¬eof(input) do
   begin S(ch); read(ch)
   end
```

must be changed to

```
while ¬eof(input) do
   begin read(ch); S(ch)
   end
```

Some welcome consequences are:

- eof(f) is always defined. (It is always true when a file is being generated; it is true after resetting an empty file.)
- eof(input) must be false before calling read(x) . Note that read(x) now stands for

  x := input↑; get(input)

- if read(x) is called with x being of type integer or real, then the next call of read(y) - with y of type char - yields the character immediately following the number x (PASCAL 6000).

3. Packed data representation

Packed records and packed arrays are formally introduced to allow for a choice of internal data representation and storage economy. Such packing has no effect on the meaning of a program, and is achieved by insertion of the symbol packed in front of the symbol record or array in the declaration of the variable.

(Note, however, that the present compiler ignores the symbol packed in the case of arrays, and that it is generally recommended that indexed access to components of packed arrays will not be permitted.)

4. The class structure

The class structure is eliminated and pointer variables are directly bound to a type T instead of to a class variable with components of type T . Thus,

```
type P = ↑c;   var c: class n of T
```

is replaced by the declaration

```
type P = ↑T
```

5. Syntactic changes

- Constant definitions are separated by semicolons instead of commas. Example:

```
const n = 10; pi = 3.14159;
```

- The former powerset structure is now called 'set structure', and the single symbol powerset is replaced by the two symbols set of.
- Labels in case statements (and variant record declarations) are separated by commas instead of colons. Example:

```
case k of
  1,3,6: A;
  2,4,5: B
end
```

- The standard function int(ch) is renamed ord(ch) and denotes the ordinal number of the character ch in the standard type char .
- The standard procedure alloc(p) is renamed new(p) .

Additional changes in PASCAL 6000

1. The procedures write and text

Revised PASCAL introduces the notion of strings as constants of types defined as packed arrays of characters. They represent a generalization of the type alfa (whose size is a machine dependent entity). Since packed arrays are not implemented in the compiler version of Dec. 72, the type alfa is still present as before. However, strings of arbitrary length may occur as parameters to the write procedure, and then will be copied onto the standard file output in full length. The procedure text therefore becomes superfluous. Example:

```
write(' THIS IS A STRING TO BE PRINTED', x, eol)
```

Note that the identifier text now denotes the standard type

```
type text = file of char
```

2. The standard file INPUT

In order to become compatible with the conventions of CDC's systems, the PASCAL file INPUT is defined as the single "logical record" (SCOPE terminology) following the program. As a consequence it has become necessary to require an EOR-card between the program and the data cards.

Note: this applies only, if the PASCAL file INPUT is also the SCOPE file INPUT (i.e. if no D-parameter is specified on the PASCAL call card).

Reference to literature

A textbook for an introductory course on programming, based on the PASCAL notation, is due to appear in April 1973:

N. Wirth, "Systematic Programming", Prentice-Hall (1973).

The German version appeared in 1972 (Teubner-Verlag, Stuttgart)

A set of solution programs to the exercises will probably also be made available.

Moreover, a "PASCAL user's manual" is in preparation.

Distribution of the PASCAL system

The PASCAL system is available for a nominal charge for tape handling, postage, and documentation from

Mr. U. Ammann
Fachgruppe Computer-Wissenschaften
ETH
Clausiusstrasse 55
CH-8006 Zürich (charge SFr. 75.--)

Dr. L.B. Smith
Computing Center
University of Colorado
Boulder, Colorado 80302
USA (charge $ 20.--)

(Only minitapes ($\leq$ 600 ft) should be sent to avoid customs handling fees.)

Compiler-Control Instructions in PASCAL 6000

Instructions controlling various modes of compilation may be inserted in the form of comments at any place in the program. Any comment is recognised as such an instruction, if the character immediately following the opening brace is a $ symbol.

{$<instr-1>,<instr-2>...<instr-n> <comment>}

Each instruction consists of a command letter followed by a + sign activating the corresponding mode, or a - sign deactivating the mode. The available command letters and modes are:

A for each assignment to a subrange variable, compile instructions which check (at run-time) whether the value assigned lies within the specified subrange.

X for each array variable, compile instructions which check (at run-time) whether the indicated index lies within the specified array bounds.

D for each division, check whether the divisor is different from zero.

C at the end of each procedure or function declaration, print the generated code.

R use rounding floating-point instructions (RX instead of FX).

The default values for these options are:

{$A+,X+,D+,C-,R+}

CU Modifications

```
*IDENT POSYS1        GHR                  73/01/30
*/          ****     ADAPT POSYS TO KRONOS.
*DECK       POSYS
*D,181,182
 ABTREQ     VFD      60/3LABT             ABORT REQUEST
 CPMREQ     VFD      24/4LCPMP,12/4,24/ERRRTN  ERROR RETURN ADDRESS

*  OUTPUT FET POINTER (MOVED TO RA+2):
 OUTFPTR    VFD      42/6LOUTPUT,18/OUTFET
*I,292
            SX2      3
            LX2      24
            BX6      X2+X6                ADD MESSAGE TO JOB DAYFILE ONLY FLAG
*I,764
            SA1      OUTFPTR              MOVE OUTPUT FET POINTER
            BX6      X1
            SA6      B1+B1                TO RA+2
            SA1      OUTFET
            SX6      4                    INSURE WRITE BIT
            BX6      X1+X6
            SA6      A1
*D,2354,2355
*  SET ERROR EXIT ADDRESS:
            SA1      CPMREQ
*I,2359
            SA4      64B                  SET ZERO WORD AT END OF ARGUMENT LIST
            MX6      0
            SA6      X4+2
            MX4      42                   MASK FOR CHECKING KEY WORDS
*I,2364
            BX0      X0*X4
*D,2597
 POS2.L2    SX6      142B                 OPEN/REWIND FOR READ
*D,2793,2797
*I,2799
 ERRRTN     BSS      0
*D,2801,2803
            SA5      B0                   FETCH RA
            MX1      54
            LX5      36
            BX1      -X1*X5               EXTRACT ERROR CODE
            SX1      X1-2                 ERROR 2 = MODE ERROR
            LX5      54
            SX5      X5                   EXTRACT ADDRESS OF MODE ERROR
*D,2852
*D,2854,2856
*D,2858,2862
*EDIT POSYS
```

```
*IDENT POSYS2       GHR                 73/01/30
*/        ****      REMOVE FREQUENCY COUNT FEATURE.
*DECK     POSYS
*D,167,168
*D,186
*D,217,220
*D,236
*D,382,405
*D,596,746
*D,751,752
*  UPON ENTRY, B3 = VALUE TO STORE AT EXECFLAG.
*D,754
*D,791,804
*D,812
*D,814,826
*D,2498,2510
 POS1.L7  SB7       OUTFET-1
*D,2564,2565
 POS1.L16 RJ        GO
*D,2593
*D,2712,2713
*  EXECUTE THE LOADED PROGRAM:
*D,2719,2764
 POS3.L16 SB3       B1+B1               FOR [EXECFLAG]
          RJ        GO
*D,2893
          NZ        X1,ABN.EX3
*D,2902,2905
*EDIT POSYS
```

```
*IDENT POSYS3       GHR                 73/02/06
*/          ****    ABORT JOB ON SYNTACTICAL ERRORS.
*DECK       POSYS
*D,2582,2583
*  IF SYNTACTICAL ERRORS WERE FOUND, PRINT ERROR MESSAGES
*  AND ABORT JOB:
*D,2594
            EQ      ABN.EXP
*EDIT POSYS
```

```
*IDENT POSYS4          GHR                 73/02/06
*/          ****       PRINT PMD ERROR MESSAGE ON OUTPUT AS WELL AS DAYFILE.
*/          ****       SEE ALSO PMD1.
*DECK       POSYS
*I,1011
 SAVEMSG    BSS2       1
*I,1071
            SA4        SAVEMSG             SET ERROR MESSAGE ADDRESS
*I,2863
            SX6        X0
            SA6        SAVEMSG             SAVE ERROR MESSAGE ADDRESS
*EDIT POSYS
```

```
*IDENT PASCAL1      GHR                 73/01/30
*/        ****      REPAIR CARDS TRUNCATED AT 72 COLUMNS BY MODIFY.
*DECK     PASCAL
*I,1731
          ;
*D,1875
            IF (GATTR.TYPTR↑.FORM = NUMERIC) ∧
               (LFP↑.VTYPE = REALPTR)
            THEN
*D,2024
                                                        ELSE JUMPTO(PUTB);
*D,2248
                  IF (GATTR.TYPTR↑.FORM = NUMERIC) ∧
                     (GATTR.TYPTR ≠ INTPTR)
*D,3614
                    IF CTPTR↑.FORM IN [CLASSS.FILES]
                    THEN ERROR(24);
*I,4193
      ;
*EDIT PASCAL
```

```
*IDENT PASCAL2      GHR                 73/01/30
*/        ****      ADD ALPHABETIC TOKENS EQUIVALENT TO SPECIAL SYMBOLS.
*DECK     PASCAL
*D,249,250
      WD : ARRAY [0..41] OF ALFA;
      WNO,WCL : ARRAY [0..41] OF SHRTINT;
*D,260,261
      WD = (=IF=,=DO=,=TO=,=OF=,=IN=,=OR=,=LT=,=LE=,=GE=,=GT=,=NE=,=EQ=,
         =END=,=NIL=,=FOR=,=DIV=,=MOD=,=VAR=,=SET=,=AND=,=NOT=,
*D,287,288
      WNO = (23,31,33,27,8,7,6*8,
         22,36,32,6,6,43,38,6,5,
*D,293,294
      WCL = (0,0,1,0,7,3,1,2,3,4,5,6,
         0,0,0,4,5,0,5,3,1,
*D,299
      WL = (0,0,0,12,21,28,36,40,40,41,42,42);
*EDIT PASCAL
```

```
*IDENT PMD1          GHR                 73/02/06
*/          ****     PRINT PMD ERROR MESSAGE ON OUTPUT AS WELL AS DAYFILE.
*/          ****     SEE ALSO POSYS4.
*DECK       PMD
*I,9
*           X4 = ADDRESS OF ERROR MESSAGE SENT TO DAYFILE.
*I,111
            SX7      X4
            SA7      SAVEMSG            SAVE ERROR MESSAGE ADDRESS
*I,122
            SA1      SAVEMSG
            ZR       X1,NO              IF NO MESSAGE
            SX1      1R
            SX2      1
            CALL     CHAR               SEND BLANK FOR CC
            SA1      SAVEMSG
            CALL     TEXT               SEND MESSAGE
            NLINE                       NEW LINE
 NO         BSS      0
*I,205
 SAVEMSG    BSS      1
*EDIT PMD
```

UNIVERSITY OF COLORADO
BOULDER, COLORADO 80302

COMPUTING CENTER

11 September 1973

PASCAL DISTRIBUTION BY
UNIVERSITY OF COLORADO COMPUTING CENTER

Contact: George H. Richmond
Computing Center, RB#3
University of Colorado
Boulder, Colorado
(303) ~~443-2211~~, extension 8131

492-8137
6934

Distribution includes a magnetic tape containing the latest system (as released from ETH in Zürich) and a package of documentation which includes those items shown in the following list.

Distribution costs to any point in North America are:

$30.00 for the complete package including a new magnetic tape
$20.00 for documentation package and writing on a supplied tape
$20.00 for the complete package and tape to previous recipients
$10.00 for documentation and writing on tape for previous recipients

The distribution costs may be paid by sending a check (payable to the University of Colorado) to the above contact or by having an invoice sent to the requestor.

Documentation (tape of December 15, 1972) includes:

- The PASCAL Distribution Tape (1 p.)
- Notes Accompanying the PASCAL Tape of December 15, 1972 (8 pp.)
- A note to Users of the PASCAL Language (6 pp.)
- CU Modifications (7 pp.)
- Planned Changes to the Programming Language PASCAL (7 pp.)
- Changes to the Programming Language PASCAL (8 pp.)
- The PASCAL Operating System POSYS (Obsolete) (20 pp.)
- Advantages of the Value Parameter (VP) over the Constant Parameter (CP) (3 pp.)
- Run-Time Check Options (1 p.)
- How to Use the PASCAL 6000 System (3 pp.)
- The Standard Procedure WRITE followed by Post-Mortem Dump (PMD) (4 pp.)
- "The Programming Language PASCAL (Revised Report)" by Niklaus Wirth (53 pp.)
- "An Axiomatic Definition of the Programming Language PASCAL" by C.A.R. Hoare and N. Wirth (32 pp.)
- The PASCAL System, Documentation, and Literature (2 pp.)

UNIVERSITY OF COLORADO
BOULDER, COLORADO 80302

COMPUTING CENTER

The PASCAL Distribution Tape

The magnetic tape distributed by the University of Colorado is a 556 bpi unlabeled SCØPE 3.2 internal binary tape, compatible with the standard KRØNØS binary format, with the following contents:

FILE	CONTENTS
1	Card image source of the unmodified PASCAL system as received from N. Wirth. See "Notes Accompanying the PASCAL Tape of December 15, 1972" for a complete description.
2	Binary decks of the PASCAL system. See reference above.
3	A MØDIFY ØPL derived from File 1 above and modified as indicated in "CU Modifications".
4	Binary decks of the PASCAL system as run at the University of Colorado from the source in File 3.
5-8	A duplicate copy of Files 1 thru 4.