Planned changes to the programming language PASCAL

N. Wirth, June 1972

The programming language PASCAL has now been in use at ETH since two years. During this time, the language has been extended by a few features - packed records, external files, read and write procedures - and the compiler has undergone many improvements. However, there have been practically no changes; the language has been kept as stable as possible.

With two years experience in the use of PASCAL, we now contemplate to introduce a few features which cannot be classified as mere extensions but which will cause some true changes in the language. The purpose of this note is to inform the users of PASCAL of our intentions before the fact. We briefly explain the new facilities and summarise the changes which must be made in existing programs. We will try to give a motivation to these changes; however, it is not possible to describe the entire set of arguments in extenso within this short communication.

I. Files

The present concept of sequential files lacks in clarity and simplicity of definition. It is for instance difficult to explain the exact role of the buffer element and the value of the function eof . We therefore adopt a definition based on the following notation:

The concatenation of two sequences $f = \langle x_1, x_2 \ldots x_m \rangle$ and $g = \langle y_1, y_2 \ldots y_n \rangle$ is denoted by $f*g = \langle x_1, x_2 \ldots x_m, y_1, y_2 \ldots y_n \rangle$ The empty sequence is written as $\langle\ \rangle$.

We regard every file f as being the concatenation of two parts (say left and right of the reading head) denoted by $\overleftarrow{f}$ and $\overrightarrow{f}$ such that always

$$f = \overleftarrow{f} * \overrightarrow{f}$$

We denote a file f as a triple $f = [\overleftarrow{f}, f\uparrow, \overrightarrow{f}]$, consisting of left part, buffer element, and right part. The initial value of a file is $[\langle\ \rangle, ?, \langle\ \rangle]$, unless it is declared as an external input file. In Pascal 6000, this is described as

```
var f[in]: F
```

In this case, the initial value is $f = [\langle\ \rangle, first(f_0), f_0]$, where first(f) denotes the first component of f . The basic file operators are now described as follows:

1. eof(f) : the right part is empty.

   eof(f) = ($\vec{f}$=< >)

   Notice that eof is always defined, not only after executing a get operation.

2. put(f) : append an element to f .

   {f = [a,x,< >]} put(f) {f = [a*<x>,?,< >]}

   Notice that the effect of putting is only defined, if eof(f) .

3. reset(f) : resetting f to the beginning for reading.

   {f = [a,?,b]} reset(f) {f = [< >,first(a*b),a*b]}

   Notice that f↑ is only defined, if a*b≠< >, i.e. if ¬eof(f) .

4. get(f) : pick the next element of f .

   {f = [a,?,<x>*b]} get(f) {f = [a*<x>,first(b),b]}

   Notice that f↑ is defined only if ¬eof(f) .

5. rewrite(f) : a new procedure with the effect

   f := [< >,?,< >]

   It is used to discard the current value of f .

6. read(ch) is equivalent to

   ch := input↑; get(input)

7. write(ch) is equivalent to

   output↑ := ch; put(output)

The essential change with respect to the existing file system is that the reset operation implies a subsequent get operation. Also, an external input file declaration (including the standard file "input") implies a first get operation. Sequential reading of the file input was programmed as follows:

```
get(input);
while ¬eof(input) do
    begin P(input↑); get(input)
    end
```

or

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

In the first case, the first line can simply be deleted under the new scheme. However, the second version must be reprogrammed as

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

In PASCAL 6000, the read procedure has been extended to also accept arguments of type integer and real. The effect of the change is that in the sequence of statements

```
read(x); read(ch)
```

ch is now assigned the character immediately following the character sequence representing the number x .

II. Packed arrays

The introduction of packed arrays, declared by preceding an array definition by the symbol packed, is in the first instance a pure extension of PASCAL, and may be implemented on machines with small wordlength (bytes) by simply ignoring the symbol packed. However, the intention is to treat the type alfa as defined by the standard declaration

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

A problem arises because now we have an instance of structured constants in the language: the alfa constants are literals of an array type. We will call them strings. Since there may be packed array variables of various dimensions, an automatic padding of strings to length 10 becomes illogical. And herein lies the change induced by the introduction of packed arrays:

1. a string with n characters is taken to be a constant of type

   ```
   packed array[1..n] of char
   ```

2. assignments of packed arrays are subject to strict rules of type compatibility, possibly with the exception that trailing blanks in literal strings may be omitted and are inserted by the compiler.

3. if a string occurs as parameter of a write statement, it is printed literally and without padded blanks. The PASCAL 6000 procedure "text" becomes redundant.

III. Parameters of procedures

The definition of constant-parameters will be changed. In every case the value of the actual parameter will be assigned to the formal parameter which effectively constitutes a local variable. Assignments to this local variable are then no longer prohibited; the parameter essentially corresponds to Algol's value parameter. In place of the symbol const we therefore will require the symbol value (which, however, may be elided).

This replacement of constant by value parameters will actually not require any changes to existing correct programs. However, the failure of the existing compiler to point out assignments to structured constant parameters has caused that many PASCAL programs are assumed to be correct although they are not. These programs may very likely produce different results under the scheme with value parameters. An example follows:

```
type  A = array[1..10]  of  integer;
procedure  P(x: A);
  begin ... x[3] := 17; ...
  end;
P(a)
```

The assignment to x[3] is illegal, if x is considered as a constant parameter. The current compiler version, however, will assign 17 to a[3] . In case of a value parameter interpretation, the assignment will be made to the local x[3] instead, and a[3] will remain unchanged.

IV. Classes and pointers

The reason for introducing the class variable is to make a dynamic allocation scheme and access of elements via pointers available, and at the same time to obtain a cheap storage retrieval mechanism for classes that are no longer needed. Such classes which have a limited lifetime as a whole are declared local to a procedure and are therefore "released" by the normal stack allocation scheme upon exit of the procedure. However, the benefits of this scheme turn out to be hardly ever used: class variables are declared in the main program almost without exception. In this case, the following simplification - which amounts to the elimination of the class variable alltogether - becomes attractive:

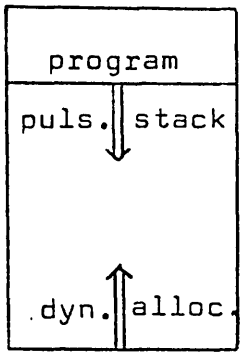
Instead of

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

we write only

```
type  P = ↑T;
```

The natural technique to handle dynamic allocations in this case is the following:



It is planned that the described changes will be implemented in a compiler to be released in late 1972. However, we reserve the rights to deviate from these descriptions, where this turns out to be necessary or desirable. At ETH Zürich, a compiler version to be put into operation on July 15 will already include change I on files. The current compiler, however, will remain available at least until summer 1973.