A Note on Dynamic Arrays in PASCAL

B. J. MacLennan
Computer Science Department
Purdue University
West Lafayette, Indiana 47907

PASCAL is frequently criticized for its lack of any variety of dynamic array facility. This lack is particularly unfortunate for systems programs which must manipulate activation records and segments whose sizes are not known at compile time.

PASCAL has been carefully designed so that all structures (except files) are static. This allows the compiler to generate efficient code by determining all offsets at compile time. Furthermore, it eliminates many of the run time checks that would otherwise be necessary to accomodate PASCAL's "strong" type system. It is claimed that the extension to PASCAL proposed below will not contradict these goals and is in general consistent with the spirit of the language. It is hoped that it will generate lively discussion.

The variant record type in PASCAL already has some of the dynamic characteristics we require. Consider the declarations:

```
type vrec = record
  head: char;
  case n: 1..3 of
    1: (rv: real);
    2: (cv: char);
    3: (iv: integer)  end;
var v: vrec; p: ↑vrec;
```

The variable v will have to be allocated enough storage for any value of type vrec, i.e. there must be space for the largest variant. If a new cell of type vrec is to be created, its reference can be assigned to pointer p by

```
new (p, x);
```

where x has value 1, 2 or 3. The effect is to create a cell of sufficient size to hold a type x variant of vrec, which might be considerably smaller than the maximum possible vrec. In other words, the size of the cell is determined at allocation rather than compile time.

Consider the following variant record declaration:

```
type vrec = record
  head: char;
  case n: 1..3 of
    1: (A1: array [1..1] of char);
    2: (A2: array [1..2] of char);
    3: (A3: array [1..3] of char)  end;
```

This declares a record with a constant size "head" and a variable length array "tail". What we are proposing is that this example be abbreviatable as:

```
type vrec = record
  head: char;
  case n: 1..3 of varying
    A: array [1..3] of char  end;
```

As before the variant record represents a finite union of types which are discriminated at run time by a "tag" field.

As an additional example, consider this declaration of strings (character sequences):

```
type string = record
  case length: 0..1000000 of
  varying val: array [1..1000000] of char
end;
```

Of course we would never want to declare a variable to be of type string - it would consume an unreasonable amount of storage. If p were of type ↑string (pointer to strings) then

```
new (p, 25);
```

would allocate storage for a string of length 25. As expected, the length of string p can be interrogated with p↑.length.

Specifically, it is being suggested that PASCAL admit record declarations of the form:

```
record <field-list>
  case <var1>: <type1> of
  varying <var2>: <type2>  end
```

where <type$_2$> must be an array type whose index type is a subrange of <type$_1$>. In all cases except allocation, <var$_2$> will act like a static array of type <type$_2$>.

As with other variant record types, it is the programmer's responsibility to maintain the tag field. It will be desirable in all cases to have a compiler option which compiles code to check the tag field for consistency at each reference to the variant part of the record.

In summary, this extension to PASCAL provides the facility of allocating cells whose sizes are determined at run time. It does this with a minimal change to the syntax and without invalidating PASCAL's strong type system. This last characteristic results from the fact that the proposed notation is little more than a macro extension of the variant record facility.

Comment on A Note on Dynamic Arrays in PASCAL

N. Wirth, ETH Zürich, Switzerland

In his recent contribution B.J. MacLennan hopes to generate a lively discussion on a proposal to introduce dynamic arrays into the language PASCAL [1] . As designer of this language I feel particularly challenged to comment.

The absence of dynamic arrays is clearly the most frequently cited shortcoming of PASCAL. Both disadvantages and benefits of this lack have been expounded before and need not be discussed here [2,3]. It is clear that a simple and cheap means of introducing dynamic arrays when needed would be most welcome. Hence, Mr. MacLennan's attempt is certainly well motivated. It also tackles the problem - and the language - at the one place that is most likely to yield success, namely where dynamic allocation is provided. Yet, I must admit reservation about the particular "solution" presented. It epitomizes the art of language grafting, and with due respect for the cleverness of the grafter I dare to point out some misconceptions underlying this art.

The indicated solution to the array problem is natural, even evident, to the professional PASCAL programmer, because he has learned to see the implementation of the various facilities behind their facade. However, to the programmer dealing exclusively with the language's high-level abstractions, the proposed formulation appears as highly artificial and unmotivated. To him the reason for this choice of notation for dynamic arrays are obscure; the virtues of a high-level language are tarnished and its purpose is compromised.

A second reservation against the proposed solution is that it suggests generality where there is none. The variant record declaration offers many more constructions than would be meaningful when declaring a "varying" component.

Perhaps most important is the fact that introduction of dynamic arrays in the language PASCAL presents no problems at all; merely admit expressions instead of constants only in the bound specifications of array declarations. But what Mr. MacLennan (and others) have tried to achieve is the incorporation of dynamic arrays in their PASCAL compiler in the cheapest possible way. Perhaps such solutions, although valuable in the context of a particular project, should not be considered as general extensions of a language, but rather as what they are: fixes to achieve some desired effect in an expeditious way.

In order to end in a positive note, let me propose a compromise that should satisfy the man in need and at the same time avoid deleterious effects on the high-level character of the language.

1. Introduce a new construct that can be used in conjunction with the definition of a pointer type only:
   type T = ↑row of T_o
2. Extend the procedure new such that it allows the specification of a row length n for such types:
   new(t,n)
3. Introduce the functions length applicable to such rows:
   length(t)
4. Allow indexing of "rows":
   t↑[i] $1 \leq i \leq n$

(Evidently, one might introduce the two array index bounds instead of the length; use of array instead of row would then be appropriate.) The obvious representation of such a row would be as a record whose first field contains the (unchangeable) length (or index bounds), and whose second field represents the array with elements of type T_o. This compromise shares with all other proposals the drawback that it extends rather than simplifies an already sufficiently complex language. It should therefore be followed only after careful deliberation.

References

1. B.J. MacLennan, "A note on dynamic arrays in PASCAL", SIGPLAN Notices 10, 9, 39-40 (Sept. 1975)

2. O. Lecarme and P. Desjardins, "Reply to a paper by A.N. Habermann on the programming language PASCAL", SIGPLAN Notices 9, 10, 21-27 (Oct. 1974)

3. N. Wirth, "An assessment of the programming language PASCAL", IEEE TSE, 1, 2, 192-198 (June 1975).