SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 9, 839–841 (1979)

Implementing Strings in Pascal—Again

ARTHUR SALE
Department of Information Science, University of Tasmania, Tasmania

SUMMARY

Following an earlier proposal that strings be realized in Pascal using the existing sequence abstraction, two simulating implementations which support the concept via a package have been written. This short paper addresses two important questions regarding such implementations which are not fully covered by Bishop's paper.[3]

FULL VS PARTIAL IMPLEMENTATION OF STRINGS

In an earlier paper,[1] it was proposed that Pascal compilers could be modified to permit a type string, defined as

type *string* = **file of** *char*

with some relaxation of the restrictions placed on file types by the Pascal Report.[2] This proposal prompted two Pascal users to write packages, consisting of procedures, functions and declarations, which could be included in programs and run under existing compilers to achieve a nearly equivalent effect. In both cases[3, 4] an important aim was the possibility of simply altering the syntax of programs written to use a package to run equivalently under a real string implementation.

There are two serious problems with such package implementations which should be emphasized. This note aims to bring them to the attention of users, in the hope that the problem areas will be detoured in practice, rather than being subtly used with consequent loss of portability.

UNDEFINITION OF FILE STATE

In the earlier paper, a distinction was drawn between the sequence abstraction itself (the conceptual data structure) and its realization in Pascal as the file structuring method. A Pascal file has, in addition to its sequence, additional data known as the file buffer variable and the file state. These allow Pascal programs to move efficiently through the sequence accessing or constructing one component at a time. However, I argue that these accretions are not intrinsic to the sequence concept, but are part of a particular access method effectively attached to a file by the execution of a *reset* or *rewrite* operation.

Obviously, it is possible to think of a sequence object as having a value without any defined buffer or state, and strings of characters are an obvious example. However, in introducing strings into Pascal implementations, it would create an inconsistency to disallow these features for one particular type of Pascal file. Consequently, the definition of string operations was earlier stated to leave the value of the file buffer and the file state (read or write) undefined.

0038–0644/79/1009/0839$01.00 *Received 4 January 1979*

Tennent[5] has argued that the file status ought to be completely defined after a whole-sequence operation, equivalent to a *reset* if the string has simply been accessed, and positioned at end-of-file in write-status if the string has been created. Treating these attributes in this way creates consistency problems for comparisons and other operations. For example, comparison of two strings should imply comparison of the two sequences (and is the obvious interpretation if the accessing is seen as 'separate' from the abstraction), whereas the acceptance of attributes as an intrinsic part of the file structure implies that two files may only be equal if their buffers and file-states are equal, with some doubt as to whether the whole of the sequences are to be compared or from the current position to end-of-file, depending on the axiomatization chosen.

The implications of this distinction for users of string packages are that calls to package procedures should not be interleaved with sequential accessing unless separated by intervening *reset* or *rewrite* calls. Fortunately, most users of string facilities will only want to call the string manipulation procedures, and whatever definition a particular package gives to the access attributes will then be irrelevant.

A more secure version of Bishop's package could be created by inserting undefinition statements at the close of the relevant whole-sequence procedures, i.e. *gets*, *puts*, *resets*, *rewrites* or *eofs*, for each external string variable accessed during the call. These statements should assign the value *notyet* to the file attribute *openstatus*, and an 'undefined value' to the character buffer, if such a value exists. Otherwise a space could be used; it already substitutes for 'undefined value' in the buffer-variable of text files positioned at end-of-line.

Two allied insecurities still exist . . .

ASSIGNMENT AND VALUE PARAMETERS

Since writers of string packages use existing Pascal structuring to create a simulation of a string type, it follows that assignment is a legal operation upon whatever structure represents a string variable.

Bishop uses a record which is an extended string descriptor to represent a string variable, with text chunks chained off it. Consequently, the unwary user of this package who writes

```
s1 := s2
```

instead of

```
assign (s1,s2)
```

acquires instead a second copy of the descriptor. The strings *s*1 and *s*2 are now linked, and operations on *s*1 may affect (a) the new descriptor and (b) the text chunks themselves. The net result is an appalling mess.

An improvement which does not remove the problem, but limits the mess to controllable proportions, is to represent a string variable by a pointer to a descriptor:

```
type string = ↑record
                s: char;  {window}
                start, current: ↑chunk;
                length, position, chunkno: natural;
                openstatus: (forreading, forwriting, notyet)
              end;
```

Accidental assignments then simply copy the pointer. The two strings are still linked, but only one descriptor will exist (unless a diabolical package subverter deliberately manufactures another). The package procedures will naturally require minor alterations, and access to the file buffer will become *s*1 ↑.*s* instead of *s*1.*s* in the original package.

With care, accidental string assignments can be avoided by users, and the assign procedure is available to achieve the desired purpose. However, if a string variable is used as an actual parameter to a formal value parameter, the same problem recurs in more insidious guise. Since the assignment in a value parameter is implicit, there is absolutely no way that a package can simulate the value-parameter passing mechanism for a string so as to hide the differences between the simulation and the real thing. (A preprocessor could handle it, of course.)

Bishop suggests that value parameters are useful with her package, but gives an example of a function internal to the package which could be rewritten. Also, the argument relies on the function given (*length*) leaving the file attributes discussed earlier untouched. Since *length* is a whole-sequence operation on a string, it ought to leave the file attributes undefined, as argued earlier.

Since there is little practical use for value parameters when strings are simulated (rather than fully implemented) and serious insecurity, I suggest that

(a) Users of string packages never use the value mechanism for strings, and

(b) the pointer-to-descriptor implementation be preferred.

The first recommendation will enhance portability, and allow for transference to a full implementation of strings. The second recommendation eliminates the possibility of disagreeing descriptors, and makes both parameter mechanisms have an identical effect for simulated strings, provided that the user does not tamper with the pointer values deliberately. Any slips can then be patched up later without altering the effect.

REFERENCES

1. A. H. J. Sale, 'Strings and the sequence abstraction in Pascal', *Software–Practice and Experience*, **9**, 671–683 (1979).
2. K. Jensen and N. Wirth, *Pascal User Manual and Report*, 2nd edn., Springer-Verlag, Berlin, 1972.
3. J. M. Bishop, 'Implementing strings in Pascal', *Software–Practice and Experience*, **9**, 779–788 (1979).
4. J. S. Parry, 'The Pascal string library notes', *Information Science Student Report*, University of Tasmania, 1978.
5. R. Tennent, private communication.