A String Extension for Pascal

F. D. Boswell
M. J. Carmody
T. R. Grove

Computer Systems Group
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

Abstract

The Pascal language, as described by the ISO Draft Proposal or Jensen and Wirth, is deficient in its facilities for character-string manipulation. A proposal for an extension to Pascal which facilitates the manipulation of character strings is given. The extension consists of a modification to the "type compatibility" rules of the language, plus a set of pre-defined functions and procedures. The extension has been implemented in the Waterloo Pascal compiler and the Waterloo microPascal interpreter.

Introduction

The Pascal language, as defined by the ISO Third Draft Proposal[ISO81] or Jensen and Wirth[Jen74], makes the manipulation of character strings inconvenient. An extension to Pascal which provides convenient variable-length string manipulation has been designed and implemented. Three factors influenced the design of the extension:

- Wirth's objective of a "reliable and efficient" implementation;
- observation of other Pascal string extensions (notably the Berkeley Pascal for Unix†[Ker81]); and
- observation of the string manipulation facilities in other languages (such as C[Ker78] and WSL[Bos82]).

These influences lead to an extension which causes a minimal perturbation of the "standard" language, and which, at the same time, is reasonably concise and efficient. Additionally, the extension has been designed in such a way as to allow standard-conforming Pascal programs to behave as always, without interference.

The string extension described here (the "Waterloo Pascal string extension") has been implemented in the Waterloo Pascal compiler (IBM 370) and the Waterloo microPascal interpreter (IBM VM/370 CMS; IBM Personal Computer; microWAT; Commodore SuperPET).

The Problem With Standard Pascal Strings

Standard Pascal defines a string type to be a **packed array** [1 .. *n*] **of** *char* (where *n* is the length of the string). This definition leads to four fundamental problems from a practical viewpoint:

---

†Unix is a trademark of Bell Laboratories.

(1) String types are compatible only if they are of exactly the same length, making it inconvenient to render common string-processing algorithms in Pascal. For example, in standard Pascal, 'abc' could not be assigned or compared to a string variable of any length except 3.

(2) Single character constants such as 'a' are of type *char* and are therefore not compatible with any string types, even **packed array** [1 .. 1] **of** *char*.

(3) In standard Pascal, there is no way to represent the null string with a constant. The zero length string constant '' is not allowed.

(4) The reading of strings (from *text* files), while not impossible, is inconvenient due to the view of Pascal that strings are collections of single characters.

The Waterloo Pascal Extension

The extension for string processing in Waterloo Pascal provide the following facilities:

(1) type compatibility for unequal length strings;

(2) a simple and efficient mechanism for variable length strings;

(3) a method for representing string constants which are only one character long, as well as the null string;

(4) a method of reading string input; and

(5) a library of routines for manipulating strings.

The following sections provide informal descriptions of the various aspects of the string extension.

String Type Compatibility

The extension defines all string types to be uniformly compatible, without regard to their length. For example, suppose the following declarations were made:

```
var
  str1 : packed array [ 1 .. 10 ] of char;
  str2 : packed array [ 1 .. 5 ] of char;
```

When a string is assigned to a variable which is not of sufficient length, the string is truncated to the appropriate length (the length of the destination variable). This extension also applies to value parameter passing which has the same type compatibility rules as assignment. Hence, the Waterloo Pascal string extension allows the following assignment statements, which would not be allowed in standard Pascal.

```
str1 := str2;
str2 := str1;
str1 := 'abcdefg';
str2 := 'abcdefg';
```

The extension also allows the relational operators to be applied to strings of unequal length; for example:

```
str1 = str2
str2 <= 'ab'
str1 > 'abcdef'
```

String Constants

The usual Pascal representation for string constants, using single quotes, is of course supported. Note that a string constant of length 1 is given type *char*. The extension provides that strings may be enclosed in double quotes. When double quotes are used, a string constant of length 1 will be given a string type, as opposed to type *char*. The null string may be represented by two adjacent double quotes (""). The following examples illustrate these lexical extensions:

```
str1 := "";
str1 := "a";
str1 := "ab";
str1 <> ""
str1 > "a"
str1 <= "abcd"
```

A convenient convention, which is used in the remainder of this description, is to use double quotes for all string constants and reserve single quotes only for single character constants.

Variable Length Strings

Strings may be thought of as *variable length* up to some fixed maximum which is specified in their declaration. A string variable of length 10, such as *str1* declared above, may contain any string from 0 to 10 characters in length. A special character value, *StrEnd*, is used to mark the end of a string which occupies less than the maximum length of a string variable. When the assignment statement

```
str1 := "abc"
```

is executed, the individual components of variable *str1* have the following values:

```
str1[ 1 ] = 'a'
str1[ 2 ] = 'b'
str1[ 3 ] = 'c'
str1[ 4 ] = StrEnd
str1[ 5 ] is undefined
...
str1[ 10 ] is undefined
```

If *str1* were assigned a string of length 10 (or more), no *StrEnd* character would be stored in *str1*. For most applications, the Pascal programmer need not even be aware of the underlying representation that is used for the variable-length strings.

Procedures and Functions For Use With Strings

Generally, those procedures and functions which take string arguments, such as write, reset and rewrite, take the extended string parameters. In the descriptions that follow, the identifiers *string*, *str1*, *str2* and *dest* represent strings, and the identifiers *length* and *offset* represent integers.

Procedures

StrConcat(*str1*, *str2*)

Procedure *StrConcat* concatentates *str2* to *str1*. *Str1* must be a string variable. For example, if

```
str1 := "ab";
StrConcat( str1, "cd" );
```

were executed, *str1* would contain "abcd".

StrDelete(string, length, offset)

Procedure *StrDelete* removes *length* characters from *string* starting at index *offset*. *String* must be a string variable. For example:

```
str1 := "abcxxxdef";
StrDelete( str1, 3, 4 );
```

leaves *str1* containing "abcdef".

StrInsert(str1, str2, offset)

Procedure *StrInsert* inserts *str2* into *str1* at index *offset*. *Str1* must be a string variable. For example:

```
str1 := "abcdef";
StrInsert( str1, "xxx", 4 );
```

leaves *str1* containing "abcxxxdef".

SubStr(string, length, offset, dest)

Procedure *SubStr* assigns the segment of *string* starting at *offset*, of the specified *length*, to the string variable *dest*. For example:

```
str1 := "abcdef";
SubStr( str1, 3, 3, str2 );
```

leaves "cde" in *str2*.

Functions

StrLen(string) : integer

Function *StrLen* returns the length of a string. For a string constant this is the length of the string, and for a string variable it is the length of the string which currently is contained in the variable. For example:

```
StrLen ( "abc" ) = 3
```

If the following assignment were executed:

```
str1 := "abc";
```

then *StrLen(str1)* would be 3.

StrScan(*str1*, *str2*) : *integer*

Function *StrScan* searches *str1* for an occurrence of *str2*. If found, the index of the first matching character in *str1* is returned, otherwise 0 is returned to indicate no occurrence. For example:

StrScan("123abcd", "abc") = 4
StrScan("bc", "abc") = 0

StrSize(*string*) : *integer*

Function *StrSize* returns the upper bound of the index type of *string*. For a string constant this is the length of the string, and for a string variable it is the maximum length string which the variable can store.

StrSize ("abc") = 3
StrSize (*str1*) = 10

String I/O

The standard procedures *write* and *writeln* have been extended to accept variable-length string parameters. Only the used portion of a string variable (that is, up to a *StrEnd* character, if any) is written.

Similarly, the standard procedures *read* and *readln* have been extended to allow variable-length string data to be read. Characters are read into a string variable until it is full (that is, *StrSize* characters have been read), or end-of-line is encountered.

References

[Bos82] Boswell, F.D.
Waterloo Systems Language
WATFAC Publications, Waterloo, Ontario, Canada, 1982

[ISO81] The International Organization for Standardization (ISO)
Third Draft Proposal ISO/DP 7185, Specification for the Computer Programming Language Pascal
1981

[Jen74] Jensen, K. and Wirth, N.
Pascal User Manual and Report (*Second Edition*)
Springer-Verlag, New York, 1974

[Ker78] Kernighan, B.W. and Ritchie, D.M.
The C Programming Language
Prentice-Hall, Englewood Cliffs, New Jersey, 1978

[Ker81] Kernighan, B.W. and Plauger, P.J.
Software Tools in Pascal
Addison-Wesley, Reading, Mass., 1981