# AN EFFICIENT AND PORTABLE STRING PROCESSOR

by James R. Pinkert, The University of Tennessee

## ABSTRACT

In this paper the author discusses a string processing system designed to alleviate education and implementation problems normally associated with such systems. It is written as a set of FORTRAN subroutines using SAC-1 [1]. Hence, this system can be implemented almost anywhere, is easy to learn, and can be used in computer science classes with limited prerequisites.

## INTRODUCTION

String processing is an important aspect of computer programming. Consider, for example, macro preprocessors, assemblers, compilers, and natural language analyzers.

Unfortunately, there can be a problem in attempting more widespread application of string processing techniques. Some installations do not have a string language, such as SNOBOL. Others have one, but encounter limitations in allocating the time and personnel necessary to teach users the language. (In addition, users are often reluctant to devote such time.) Finally, from an educational viewpoint, the internal workings of these string languages are difficult to investigate in lower-level classes.

In this paper, the author discusses a string processing language designed to alleviate such problems. It is implemented as a set of FORTRAN subroutines employing SAC-1. The SAC-1 list processing module is an efficient, but simple, single link reference count system written in ASA standard FORTRAN. Hence, the string processing module can be implemented at almost every installation, is easy to learn, and can be discussed in classes with only a minimal background.

## STRING PROCESSOR ROUTINES

Since many of the applications of string processing involve operations on strings read in rather than generated internally, the description of the module begins with the routine to read a character string from some input device.

L=STREAD(UNIT)     String Read
Unit is the input device from which 72 characters are read. List L is a first order list of 72 cells, each of which corresponds to one of the input characters.

Many times it is desirable to have trailing blanks removed automatically from the input records. This is an option which may be activated by invoking the following routine.

TRIMON     Turn String Trimming Option On
Trailing blanks for input strings are eliminated from the input record once the trimming flag is set.

Seldom would the user want every input string trimmed of its trailing blanks. To turn the trimming option off, the user simply invokes the following routine.

> TRIMOF    Turn Trimming Option Off
> The check for trailing blanks on input strings is skipped when the trimming option flag is turned off.

The following routine outputs arbitrary length strings in records of 72 characters, padding the last record with blanks.

> STWRIT(UNIT,L)    String Write
> Records of 72 (73 with carriage control for unit 6) characters from arbitrary length list L are written to device UNIT.

Many string processing applications generate results which a user would like to save for future use. A user could write a string in 72 character blocks. However, efficiency dictates the use of the two following routines for use with input or output using mass storage devices.

> L=STPACK(M)    String Pack
> The string M is used to generate a packed list L. The number of characters packed per word is implementation dependent.

The preceding routine could also be employed to reduce the amount of space necessary to represent a string and to reduce the time to compare strings. The user should be careful, however, with string alignment within cells when using the search routines.

Naturally, a routine is provided for unpacking previously packed strings.

> L=STUNPK(M)    String Unpack
> Input is the packed list M. Output is a new, unpacked list L.

Often it is desirable to trim blanks from selected strings. Routine STTRIM provides this function

> L=STTRIM(L)    String Trimming of Trailing Blanks
> Input is the first order list L. Output is the original list with all trailing blanks removed.

The following routine provides the function of searching one string to determine if a second string is completely contained within the first string. The implemented algorithm takes time proportional to the product of the lengths of the two strings for the search. While algorithms exist for string matching which operate in time proportional to the sum of the lengths of two strings, it was believed that the overhead of such algorithms implemented in this system would have negated any possible savings in computing time.

> N=STSERC(A,B)    String Search
> Inputs are first order lists A and B. Output is the relative location in A where B is found to begin. String B must be completely contained in A.

Quite often it is necessary to determine only if two strings match symbol for symbol. Integer function STCOMP analyzes two strings and returns a one if the two strings match and a zero if they differ.

> N=STCOMP(A,B) String Compare
> Input are two strings A and B. Output is one if string A is equal to string B, and a zero otherwise.

Seldom are all the symbols of a string analyzed at any given point in time. The following routine provides the service of copying selected cells of a string so that a string may be analyzed in parts.

> L=STGETS(A,N,M) String Get a Substring
> Inputs are the string A, relative cell location $N \geq 0$, and number of cells, $M \geq 0$. Output L is a list of M cells copied from A beginning with the Nth cell of A.

The ability to put sections of strings together as desired is provided by the following routine.

> L=STREPL(A,B,N) String Replacement
> Inputs are strings A and B, and displacement N. String B is inserted in string A after the Nth cell of A. Strings A and B are nulled. Output is the combination of lists A and B.

The string processing module provides several routines which enable a user to assign one string as the value of a second string. These routines are implemented on the basis of an indirection list. The pointers to both the string and its associated value are kept in a list maintained by the system. The indirection list pointer, STNAME, is located in the labelled COMMON block, STRING. Graphically, the indirection list structure is as shown in Figure 1.

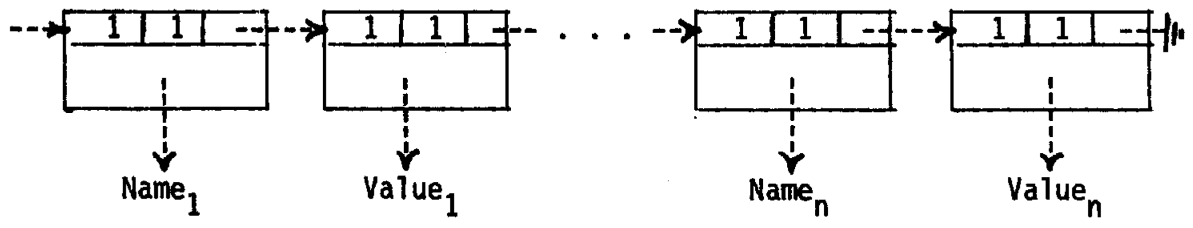

Figure 1: Indirection List

Thus, the length of STNAME is 2n for n string entries and associated values.

The string name (string entry) and its associated value are entered into the indirection list by calling the following routine.

> SSTPBS(SNAME,STRNG) Store a String Pointed To By a String
> Input are two lists, SNAME and STRNG. The input list, STRNG, is to be entered as the value of the string SNAME.

To recall the value of a string which is entered in the system indirection list, a call to RSTPBS with the string name whose value is desired is all that is necessary.

> L=RSTPBS(SNAME)  Recall the String Pointed To By a String
> Input is the string SNAME. If SNAME is entered in the indirection list, the pointer to its corresponding string value is returned. If SNAME is not entered, a "-1" is returned.

The preceding two routines require the routine to be described next. Routine STILOC returns the location in the indirection list of the pointer to the string value associated with a specified name. If there is no entry correponding to the name sent to STILOC, then a value of "-1" is returned.

Routine STILOC can be quite beneficial to the user who has reason to reference the indirection list frequently. By providing a pointer into the indirection list, and two routines to work with this pointer, the costly search time necessary when using the previous two routines can be avoided.

> L=STILOC(SNAME)  String Indirection Location
> Input is the string name, SNAME. Output is the indirection list cell location which contains the pointer to the value associated with SNAME.

If the user employs routine STILOC to get the location of the pointer to the value of a string name entered into the indirection list, then the following two routines may be used to alter or recall the value without the searching required by routines SSTPBS and RSTPBS.

> L=RSTPBP(PTR)  Recall a String Pointed To By a Pointer
> Input is the pointer into the indirection list. Output is the corresponding list value.

Although the above routines do not need to search the indirection list, there may be times when the routines SSTPBS and RSTPBS must be used. It is wise, then, to keep the size of the indirection list to a minimum. The following routine enables the user to delete entries from the indirection list when they are not longer needed.

> STIDLT(SNAME)  String Indirection Entry Delete
> Input is the string name. The string and its associated value are deleted from the indirection list.

Another useful routine for analyzing strings is a routine to remove cells from one list and to put them into a new list. This is particularly useful in analyzing patterns in strings. Once a pattern is detected, that pattern can be removed and the string analyzed for further occurrences of the same pattern or some other pattern.

> M=STRIP(L,N,K)  Strip Cells From a List
> Inputs are string L and relative cell positions $N > 0$ and $K \geq N$. Output is the list of cells N through K removed from L.

Rather than searching one string for complete containment of another string, the following routine searches for the first occurrence of any one of a set of symbols in another string.

N=STMTCH(LST,PTRN) <u>Str</u>ing <u>Match</u>
Input are two strings LST and PTRN. Output is the relative location in LST of any cell value of PTRN which matches a cell value of LST.

A routine to take two input strings and merge them into one string is STMRGE. The two input lists are nulled.

L=STMRGE(A,B) <u>String</u> <u>Merge</u>
Inputs are two strings A and B. Output is a single merged string.

## CONCLUSION

The string processing system discussed in this paper can be brought up very quickly at almost any installation having a standard FORTRAN compiler. It is easy to use, and has been successfully taught in data structure, symbolic algebra, and compiler courses. Although suitable for novices and beginning classes, it is powerful enough to do very sophisticated tasks in higher level classes and general user applications.

## REFERENCES

1. G. E. Collins, The SAC-1 List Processing System, U. of Wisconsin Computing Center Tech. Report, July, 1967.