M. D. McILROY, Editor

String Processing Techniques

Stuart E. Madnick
M.I.T. and I.B.M. Corp.† Cambridge, Mass.*

The internal organization of string processing systems is discussed. Six techniques for data structures are presented and evaluated on the basis of: (1) creation of strings; (2) examination of strings; and (3) alteration of strings. Speed of operation, storage requirements, effect on paging, and programmer convenience are also considered. One of the techniques, single-word linked blocks, is used in an example demonstrating an implementation of a SNOBOL string processing language on an IBM System/360.

1. Introduction

The title of this paper was also that of a discussion group held at a 1966 ACM Symposium on Symbolic and Algebraic Manipulation. At that time it became obvious to the author that many people are interested in developing string processing languages or utilizing string processing techniques in the solution of problems. Although there is a reasonable amont of documentation describing the external appearances of many existing string processing languages, there is a noticeable lack of information about their internal organizations.

Six string processing techniques and an example of the successful use of one of them are presented in this paper. Of course many variations of the presented techniques are possible. The IBM System/360 is used to illustrate the formats for the techniques.

2. Computer Hardware Requirements

Although this paper is basically concerned with data structures that are machine-independent, several computer hardware features are considered necessary to make meaningful use of these structures.

It is assumed that the computer main storage can be processed as words. These words can be fixed size as on the GE 635 and IBM 7094, or multiples of smaller elements as on the IBM 1620 and IBM System/360. In this paper the 32-bit word size used on the IBM 360 will be considered.

The basic symbols (characters, letters, digits) are represented by an 8-bit code called the byte. Thus 4 characters (8 bits each) can be contained in a machine word (32 bits).

It is assumed that the computer has capabilities to access and manipulate individual characters within a word. The ability to index an address is definitely desirable.

3. The Problem of String Processing

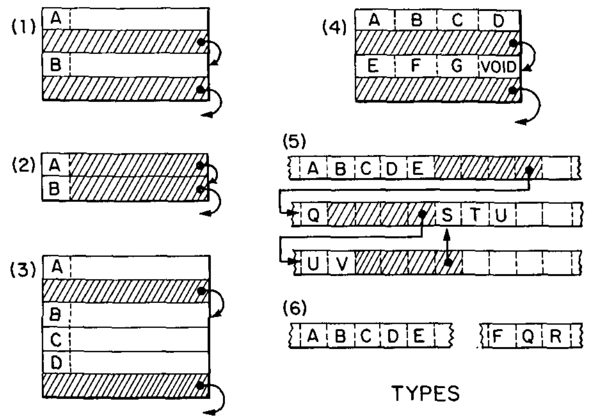
The simplest way to store a string of characters would be to put consecutive characters in successive bytes throughout memory. However, any attempt to change the length of the string results in considerable character-moving.

Most symbol-manipulating languages solve this problem by use of pointers. A pointer specifies the location of the next character on the string, thus allowing the elements of the string to be located in physically noncontiguous regions of the computer's memory and yet be logically bound together.

A pointer on the System/360 must be 24-bits long to connect string sections which are located arbitrarily in the computer. There are several symbol-pointer arrangements possible.

4. Data Structures under Consideration

Six basically different data structures with potential for numerous variations have been devised. They are described below and schematically presented in Figure 1.



(1) Double-word Blocks
(2) Single-word Blocks
(3) Variable-length Blocks
(4) Packed Double-word Blocks
(5) Linked Linear String
(6) Linear String

Fig. 1. Data structures under consideration

* MIT Electrical Engineering Department
† IBM Cambridge Scientific Center

The first four methods are based primarily upon fixed word length considerations, while the remaining two methods make use of the variable word length features.

Method 1 (*double-word blocks*). The string is represented internally by linked two-words blocks. The first word contains a character, the second contains a pointer to the next character.

Method 2 (*single-word blocks*). This method strongly resembles the double-word block technique, but, rather than using two words, the 8-bit character and 24-bit pointer are packed into a single 32-bit word.

Method 3 (*variable-length blocks*). The characters are stored one to a word consecutively in memory. Whenever the sequence is to be broken, a pointer indicates the location of the next block of characters. Characters and pointer can be identified by information stored in the unused portion of the 32-bit word.

Method 4 (*packed double-word blocks*). The characters are stored in fixed length-packed blocks (4 per word, 8 per double word, etc.) followed by a pointer to the next block. For the example presented in this paper, four characters are stored in a word followed by a pointer to the next four characters. A special character called the "void" character fills the empty spaces in data blocks that are only partially filled.

Method 5 (*linked linear strings*). Characters are stored sequentially in memory, byte by byte. Whenever the sequence is to be broken, a special character is used to denote a pointer. In other words, the pointer is made 32 bits long where the leading 8 bits identify it as a pointer.

Method 6 (*linear string*). This method can be implemented in at least two ways. The simplest (conceptually) is always to maintain strings linearly throughout memory without any pointers. An alternate scheme is to store strings linearly within large blocks (4096-characters long for example) with a pointer to the next block.

5. Storage Requirements

In discussing storage requirements the term "packing density" is used. Packing density is the percentage of storage containing character information.

Method 1 (*double-word blocks*). Since only one character is stored for every pair of words used, the packing density is only 12.5 percent. This means that at most only one out of every eight bytes of storage is used to contain data.

Method 2 (*single-word blocks*). This technique provides for a fixed packing density of 25 percent.

Method 3 (*variable-length blocks*). The packing density of this method is a function of the data processed. Initially there will be no pointers (density 25 percent), but as the data is manipulated, it may begin to resemble method 1 (density 12.5 percent). If a "garbage collector" routine is used to rearrange the data periodically into its linear structure, close to 25 percent packing density can be maintained.

Method 4 (*packed double-word blocks*). Because one to four characters will be stored for every double word used, the packing density will vary from 12.5 to 50 percent. As in method 3, a garbage collector could be used to maintain storage density.

Method 5 (*linked linear strings*). Initially all the characters will be stored linearly throughout memory. This results in a packing density of 100 percent. Under worst case conditions each character could be followed by a pointer character and a pointer, thus reducing the density to 20 percent. The use of a garbage collector to reorganize the data periodically can keep the density as close to 100 percent as desired.

Method 6 (*linear strings*). The linear storage technique results in a 100 percent packing density. Of course this method requires continual storage reorganization.

6. Speed Limitations

The ease with which certain string manipulations can be performed determines, to a large extent, the overall operating speed of a string processing application. The basic string manipulating operations are: (i) a scan (ii) an add/delete, and (iii) a storage manager or "garbage collector."

Method 1 (*double-word blocks*). Individual characters can be moved or compared either by using the character-processing capabilities, or by loading into a register and performing fixed word length operations on them.

The next element of the string can be easily accessed since the pointer is kept in the low order 24 bits of the pointer word. In this case the pointer is immediately loaded into an index register.

To delete a character or group of characters from the string, it is necessary merely to change the pointer preceding the portion to be deleted to point to the first character after the section. To add a section to the string the reverse process is used. (Before a group of characters can be inserted into the string, they must be linked together in the same form as in the string.) The pointer located on the string at the place where the insertion is to be made is moved to the bottom of the section to be inserted. It is replaced by a pointer to the first element of the new section.

There are two possible techniques that can be used to maintain free storage from which new strings are formed. One method uses a portion of available memory for stored strings, and the remainder as a bulk quantity of unused storage. A pointer keeps track of the beginning of the free storage area. As new strings are produced, the free storage is reduced. When no free storage remains, the garbage collector must move and relink the strings to create free storage from deleted elements.

Another method of maintaining free storage is to place every word of available storage on a string. This special string, called the "free string," links all unused words of storage. When sections are to be added to a regular string, the necessary number of elements is unlinked from the free string. No garbage collection is necessary since all free storage is linked together.

There is one more consideration: a multiprogramming environment with automatic paging where program segments are swapped between main memory and secondary storage. Effective use of paging requires that the data being referenced be fairly localized. In general the elements of the strings are located through memory. Since the double-word blocks method provides a pointer for every character, it is possible for each character to be located in a different section of memory. Although variations of this method, that tend to localize the string, have been developed, the additional complexity involved usually outweighs the simplicity of the basic method.

Method 2 (*single-word blocks*). This technique has the the same basic characteristics as Method 1. Since we can directly load the low-order 24 bits of the word into the index register, it is not necessary to mask off the 8 high-order bits containing the character. The only difference is a slight additional manipulation involved in the insertion of pointers without destroying the character, into the single-word block.

Method 3 (*variable-length blocks*). The characters can be manipulated by any of the methods described above. The next element of a string can be obtained by incrementing the index register, if a pointer is not present, or by loading the pointer into the index register, if the end of a block has reached. It is necessary continually to check the data to distinguish between characters and pointers.

Deletion of characters is simple. The first character to be deleted is replaced by a pointer to the character following the section to be deleted. Addition to the character string is not quite as easy. The characters to be inserted are put in consecutive words of a block obtained from free storage. The character located at the spot where the addition is to be made is moved to the top of the new block and replaced by a pointer to its new location. The last element of the new block is a pointer back to the element of the string immediately following the point of insertion.

The presence of odd-size blocks and the need for a contiguous free storage area make a garbage collector the only practical means of maintaining the storage.

Since the strings are more localized than in method 1 and 2, the variable-length block method is more practical for a computing system utilizing paging techniques.

Method 4 (*packed double-word blocks*). The characters can be removed from the packed word, byte by byte, or the entire word can be placed in a register and shifted, one character at a time. The "void" character must be detected and ignored. After all four characters have been processed, the next block of characters is reached by loading the pointer into the index register.

Deletion of characters requires several steps. Unless the first character to be deleted is at the beginning of a four letter block and the last letter to be deleted is at the end, it is necessary to "void" a number of letters in the two end blocks. Then the pointer can be adjusted to bypass the remainder of the section to be removed. To insert a section, the characters to be added are packed four to a word and linked together in the form of a string. Unless the insertion is to occur after a letter that terminates a block on the main string, the block must be separated into two blocks with the end part placed at the end of the insertion string. Unused spaces are filled with "void" characters.

Free storage can be maintained either by the use of a free storage string or a garbage collector. If the free string is used, a localized garbage collector should be used to minimize the number of "void" characters on strings.

Method 5 (*linked linear strings*). The most reasonable way to scan the linked linear string is to use the character-handling instructions. The next character is accessed by incrementing the index register or by loading a pointer into the index register. The detection and handling of the pointer must be considered.

The addition and deletion of characters is complicated. If there are four or more characters to be deleted, a pointer is placed where the first characters were located. If there are fewer than four characters to be removed, the remaining characters are moved to a block obtained from free storage, and replaced by a pointer to their new location. A return pointer is then inserted after these new characters. The insertion process is slightly more involved. The characters to be added are strung out in a block obtained from free storage. The four characters from the main string following the point of insertion are moved to the end of the new block and replaced by a pointer. Special care must be taken to check the moved characters for the presence of a pointer.

The use of a garbage collector is the only way that free storage can be maintained. The efficiency of multiprogramming is dependent upon the frequency and effectiveness of the garbage collector.

Method 6 (*linear string*). The characters on this string are trivially accessed by continually incrementing the index register.

The insertion and deletion of characters is not difficult, but it is slow.

The entire string can be recopied with the desired changes, Alternatively, to delete a section of the string, all characters to the right of the section to be deleted are moved left a number of places corresponding to the number of characters to be deleted. To add to the string, all the characters following the point of insertion are moved right the correct number of places and the new characters are inserted.

There is no need for any additional storage maintenance, since characters are always stored at 100 percent efficiency. This method is probably the most effective for operating in a multiprogramming environment, since it maximizes proximity of elements on a string.

7. Summary

No single method can be determined as "best" or "worst." Each has advantages and disadvantages; it is the application that will usually determine the most desirable method. Table I summarizes the characteristics of the six methods proposed. The results are based upon tests run on an IBM System/360.

8. Example of String Processing on the System/360

The author decided to produce for the System/360 a string processing capability similar to that of COMIT and SNOBOL. In fact, the present system is SNOBOL-compatible.

TABLE I. Data Structure Characteristics

	Packing density	*Ease of scan*	*Ease of insert delete*	*Speed of insert delete*	*Localization of strings (for paging)*
(1) Double word	12.5	easy	easy	fast	poor
(2) Single word	25	easy	easy	fast	poor
(3) Variable length	12.5–25	moderate	moderate	moderate	fair
(4) Packed double	12.5–50	moderate	difficult	slow	fair
(5) Linked linear	25–100	moderate	difficult	very slow	good
(6) Linear	100	easy	moderate	very slow	excellent

After considering the various data structures described in this paper, Method 2 (single-word blocks), was chosen. Although packing density and application for paging is considered important, speed of operation and ease of implementation was given highest priority.

Strings are defined by a three-word block called the "string reference block." The first word specifies the location of the first character on the string, the second, the ength of the string, and the third, the location of the last character on the string. Although the string contents are changed continually and rearranged throughout memory, the string reference blocks remain at fixed locations and contain the information specifying the present string contents. Figure 2 demonstrates this structure for the strings containing "CAT" and "DOG."

A set of 30 basic string processing instructions is used. They are of the form: COPY Y, APPEND Y, REPLACE Y, INSERT Y, GOTO Y, etc. A program consists of a set

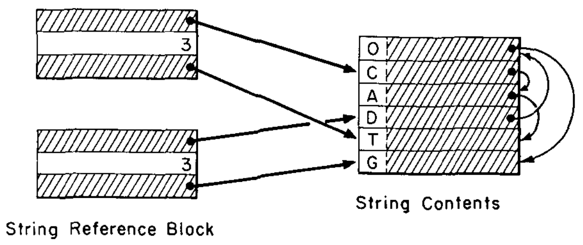


Fig. 2. String reference block structure

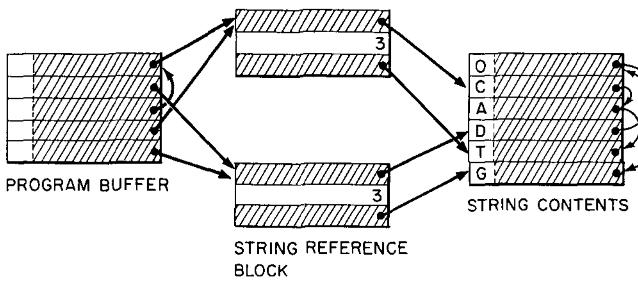


Fig. 3. Program buffer structure

of these instructions contained in the "program buffer." The program buffer is a section of memory containing consecutive 32-bit words: the first 8 bits of each word specify the instruction; the remaining 24 bits specify a string reference block in the case of a string manipulation, or the location of another instruction in the program buffer in the case of a GOTO. Figure 3 illustrates this structure.

Strings that are to be variables are assigned external names, as is the case in most programming languages. It was decided to include the ability to indirectly reference a string. To indirectly reference, rather than access a string by directly specifying its name (or reference block location), we specify the name of another string whose contents is the name of the string desired. Therefore, indirect string referencing requires a means by which the external string names (contained in a string) can be associated with the corresponding reference block location during execution. The problem of determining the reference block location of a string from its string name is further complicated by the fact that string names are of arbitrary length and may be created dynamically during execution. Of course, it is important that indirect referencing be performed as efficiently as possible. A two-step mechanism is used to solve this problem. First, the external string name is appended to the bottom of the string reference block. Second, a hash-coded symbol table is set up. The symbol table contains the relative address of the string reference block and the number of characters in the string name (see Figure 4).

Referring to Figure 4, if we want to indirectly reference string DOG (by name) through ALPHA, the letters "DOG" (the contents of ALPHA) are used as the argument of a hash function to determine the entry in the symbol table which in turn gives the location of the string reference block for DOG. Since hash functions do not necessarily produce unique results, it is necessary to compare the string name contained in the reference block indicated by the table entry with the letters "DOG". If the string names do not correspond, successive table entries are tried. In general, with a sufficiently large hash table the correct reference block is located in one or two probes.

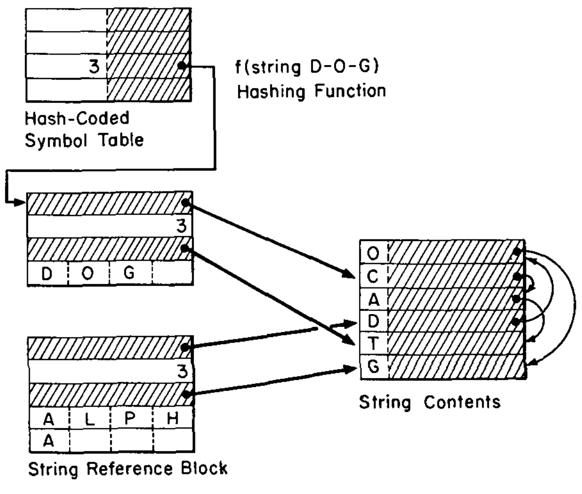


Fig. 4. Overall data structure

Although timing comparisons were not a major goal of this project, several programs were run using the above described system on an IBM 360 model 40 and compared with the SNOBOL system on the IBM 7094 which uses a linear string technique for string storage. In general, identical programs ran twice as fast on the 7094 as on the 360 model 40. Relating the speed of the two computers is difficult, but the 7094 is generally considered to be about five times faster than the 360/40 for basic operations. These results indicate the relative superiority of the single-word blocks in this case.

Acknowledgments. Special thanks are extended to Roy Harris, John Hershey, Larry-Stuart Deutsch, Payne Freret, Margaret Barovich, and Frank DeRemer for their assistance in preparing this paper. Robert Graham's advice and guidance was especially helpful.

RECEIVED MAY, 1966; REVISED MARCH, 1967

REFERENCES

1. BOBROW, DANIEL G., AND RAPHAEL, BERTRAM. A comparison of list-processing computer languages. *Comm. ACM 7*, 4 (April 1964), 231–240.
2. ——, AND WEIZENBAUM, JOSEPH. List processing and extension of language facility by embedding. *IEEE Trans. EC-13* (Aug., 1964), 395–400.
3. FARBER, D. J., GRISWOLD, R. E., AND POLANSKY, I. P. SNOBOL, a string manipulation language. *J. ACM 11* (Jan. 1964), 21–30.
4. ——, ——, AND ——. The SNOBOL programming language. *Bell Sys. Tech. J. XLV*, (July–Aug. 1966).
5. MCCARTHY, J. ET AL. *LISP 1.5 Programmers Manual.* MIT Press, Cambridge, Mass., 1963.
6. MADNICK, STUART E. SPL/1: a string processing language. MIT B.S.E.E. Thesis, June, 1966, Cambridge, Mass.
7. The MIT Computation Center. *The Compatible Time-Sharing System, A Programmers Guide.* MIT Press, 1963.
8. NEWELL, A. (ED.) *Information Processing Language-V Manual.* Prentice-Hall, Englewood Cliffs, N. J., 1961.
9. WEIZENBAUM, J. Symmetric list processor. *Comm. ACM 6*, 9 (Sept. 1963), 524–544.
10. YNGVE, V. *COMIT Programmers Reference Manual.* MIT Press, Cambridge, Mass., 1963.

Implementing Phrase-Structure Productions in PL/I

LARRY IRWIN
Ohio University, Athens, Ohio

A method is described for implementing the productions of a context-free phrase structure grammar in a PL/I procedure whose structure and statements parallel the structure and notation of the grammar.

A simple technique is described here for implementing productions in PL/I. Moreover the description becomes the driving algorithm.

Consider the following set of productions which describes the context-free phrase-structure grammar of a simplified assignment statement [1]:

$$
\begin{aligned}
&\langle\text{statement}\rangle ::= \langle\text{variable}\rangle = \langle\text{expression}\rangle\\
&\langle\text{expression}\rangle ::= \langle\text{term}\rangle \mid \langle\text{term}\rangle + \langle\text{expression}\rangle\\
&\langle\text{term}\rangle ::= \langle\text{factor}\rangle \mid \langle\text{factor}\rangle * \langle\text{term}\rangle\\
&\langle\text{factor}\rangle ::= \langle\text{variable}\rangle \mid (\langle\text{expression}\rangle)\\
&\langle\text{variable}\rangle ::= A \mid B \mid C
\end{aligned}
$$

The PL/I (F-level) procedure shown in Figure 1 produces a list of 10 arbitrary, well-formed assignment statements.

```
STMTGEN: PROCEDURE OPTIONS(MAIN);
  DECLARE (STATEMENT,EXPRESSION,TERM,FACTOR) RETURNS(CHARACTER(120)
          VARYING), VARIABLE RETURNS(CHARACTER(1));
  DECLARE CHOICE_OF /*AS THE*/ ENTRY /*NAME OF A FUNCTION WHICH ACCEPTS
          N, A*/ (FIXED BINARY) /*INTEGER VALUE, AND*/ RETURNS /*A*/
          (FIXED BINARY) /*INTEGER PSEUDO-RANDOMLY CHOSEN FROM THE
          SET (1,2,...,N)*/;

  PUT PAGE LIST('PRODUCED ASSIGNMENT STATEMENTS');
  PUT EDIT(( STATEMENT DO I = 1 TO 10 )) ( SKIP(1), A );

 STATEMENT: PROCEDURE CHARACTER(120) VARYING;
   RETURN(VARIABLE || '=' || EXPRESSION);
   END STATEMENT;

 EXPRESSION: PROCEDURE CHARACTER(120) VARYING RECURSIVE;
   DECLARE ALTERNATIVE(2) LABEL;
   GO TO ALTERNATIVE(CHOICE_OF(2));
   ALTERNATIVE(1): RETURN(TERM);
   ALTERNATIVE(2): RETURN(TERM || '+' || EXPRESSION);
   END EXPRESSION;

 TERM: PROCEDURE CHARACTER(120) VARYING RECURSIVE;
   DECLARE ALTERNATIVE(2) LABEL;
   GO TO ALTERNATIVE(CHOICE_OF(2));
   ALTERNATIVE(1): RETURN(FACTOR);
   ALTERNATIVE(2): RETURN(FACTOR || '*' || TERM);
   END TERM;

 FACTOR: PROCEDURE CHARACTER(120) VARYING RECURSIVE;
   DECLARE ALTERNATIVE(2) LABEL;
   GO TO ALTERNATIVE(CHOICE_OF(2));
   ALTERNATIVE(1): RETURN(VARIABLE);
   ALTERNATIVE(2): RETURN('(' || EXPRESSION || ')');
   END FACTOR;

 VARIABLE: PROCEDURE CHARACTER(1);
   DECLARE ALTERNATIVE(3) LABEL;
   GO TO ALTERNATIVE(CHOICE_OF(3));
   ALTERNATIVE(1): RETURN('A');
   ALTERNATIVE(2): RETURN('B');
   ALTERNATIVE(3): RETURN('C');
   END VARIABLE;

END STMTGEN;
```

FIG. 1

Thus, in general, each production becomes a function procedure which returns a varying character string and whose name is the nonterminal character on the left-hand side of the production. The right-hand side becomes the body of the procedure where each alternative in the production becomes a separate, labeled RETURN statement to be chosen at random. Characters are concatenated by the operator, ||, and immediate terminal characters are character string constants.

RECEIVED FEBRUARY, 1967; REVISED MARCH, 1967

REFERENCE

1. HULL, T. E. *Introduction to Computing.* Prentice-Hall, Englewood Cliffs, N.J., 1966.