PUC

Series: Monographs in Computer Science
and Computer Applications
Nº 09/76

IMPLEMENTING CHARACTER STRINGS IN FORTRAN

by

Rubens N. Melo

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 209 — ZC 20
Rio de Janeiro — Brasil

Series: Monographs in Computer Science
and Computer Applications
Nº 09/76

IMPLEMENTING CHARACTER STRINGS IN FORTRAN*

by

Rubens N. Melo

Series Editor: Sergio E. R. Carvalho May, 1976

* This work was supported in part by the Brazilian Government Agency FINEP under contract Nº 244/CT, and developed by the Data Base Group of the Deptº de Informática - PUC/RJ.

ABSTRACT

A simple technique for representing and manipulating variable character string is developed. Some algorithms for the main operations are described in a structured language similar to a programming language. A simple algorithm for collecting available space between strings is proposed. A concrete example in FORTRAN is also presented.

KEYWORDS

Strings, string manipulation, language extension, character, character manipulation, compaction algorithm.

RESUMO

Uma técnica simples para a representação e manipulação de strings de caracteres de tamanho variável é desenvolvida. Os algorítmos de algumas operações importantes são descritos numa linguagem estruturada similar a uma linguagem de programação. Um algorítmo simples para a compactação do armazenamento de strings é proposto. Um exemplo concreto em FORTRAN também é apresentado.

PALAVRAS CHAVE

Caracter, manipulação de caracter, extensão de linguagem, compactação.

CONTENTS

1 INTRODUCTION

The idea of implementing character string manipulation facilities in FORTRAN is not new. Most existing implementations consist of subroutine packages where strings are handled as vectors, the string description being passed as subroutine parameters [1].

Another idea is to store the string as a vector of characters but to reference it by a pair of integers (L,T) where L means the string location in Storage and T the string length. This pair is called the string descriptor and can be implemented with different techniques, for instance, as a complex number in FORTRAN [2] or a two-component vector.

In this paper the strings will also be referenced by descriptors and in the algorithms presented it is assumed that the descriptor's components are easily accessed
For clarity the algorithms are not written directly in FORTRAN but mather in a structured manner so that they can be easily understood and programmed.

Section 2 essentially shows the notation used throughout this paper . Section 3, 4 and 5 present the algorithms for some important string manipulation functions. In section 6 the storage management is discussed and an algorithm for compacting the strings representation in order to collect the available space between strings for later use. This algorithm is claimed to be better than the one described by Hanson [2] because it does not need a separate vector of pointers to sort the descriptions before compaction. Finally, the last two section give some alternatives for the implementation and present an example of use of our system.

2 STRING DESCRIPTORS

Conceptually we will work with two vectors: A vector D of descriptors and a vector B for the storage of characters. However in our implementation there will be only one vector encompassing both vectors D and B. The descriptors themselves will be nothing more than INTEGER values of FORTRAN viewed as pairs of numbers. These pairs of numbers (L,T) used throughout the algorithms are obtained by means of the functions LOC and LENGTH described in section 5.

Let B be a vector of characters

$$B = \{B_i\} \quad i = 1, 2, \ldots M$$

The value of a B_i is a character of the set C of all possibile characters in the system.

Let D be a vector of N pairs of numbers

$$D = \{D_i\} = \{(L_i, T_i)\} \quad i = 1, 2, \ldots N$$

where $1 \leq L_i \leq L_i + T_i - 1 \leq M$.

A string in B is a sequence of zero or more consecutive characters stored in B. It is described by two numbers: the index of its first character (i.e its location) and the number of its characters (i.e its length). For instance the string 'BLUE' could be stored in B_5, B_6, B_7 and B_8 and its descriptor would be (5,4).

The null string is described by length equal to zero and location equal to a valid index in B.

All the descriptors in D must be initialized as describing null strings:

$$L_i = M, \; T_i = 0 \qquad i = 1, 2, \ldots N$$

String operations such as concatenation, substring definition, etc, will be described by means of manipulations of vectors B and D.

Let d be an index of B which indicates the next available position in B. The initial value of d is clearly 1.

Let us denote by C^M the set of all string with length not greater than M which can be made out of the characters in C. A string $S \in C^M$ will be eventually denoted by $S = s_1 \; s_2 \; \ldots \; s_k$, $k \leq M$. In fact every string described by a $D_i \in D$ belongs to C^M.

Every string stored in vector B is either described by a descriptor in vector D or by the value of some string generator functions as we shall see in the next section.

3 STRING GENERATOR FUNCTIONS

This section presents some functions which compute descriptors of strings as they are stored in B. All these functions-except STRING and GETSTR - refer to the strings by their descriptors.

The Functions STRING and GETSTR

STRING ('S')

This function receives as argument the direct representation in characters of the string to be stored in B. The value returned is a descriptor for this new string.

Ex S = STRING('BLUE')

GETSTR(COL,T)

This function receives as argument a description of a string stored in a standard input file - cards for example. The first argument COL indicates the position where the string begins in the input register and the second argument T defines the string length. The function GETSTR causes the transmission of the string to the vector B and computes a descriptor for it.

The storage of a string $S = s_1 s_2 \ldots s_n \in C^M$ in B is accomplished by the following algorithm:

ALGORITHM 1

```
C   Check whether there is space available
    IF d+n > M THEN CALL COMPACT algorithm FI
C   move string S to B
    FOR J=1,n
        B(d+j-1) = s(j)
    ROF
C   Set the new value of d
        d=d+n
C   Return the descriptor
        function name = (d-n,n)
    END
```

the COMPACT algorithm mentioned above is described in section 6. The language used to describe the algorithm is a kind of a structured FORTRAN similar to the language accepted by FORTS [3]. We hope that the use of this structured language instead of pure FORTRAN improves the readability of the algorithm.

SUBSTR(D_1,I,Z)

This function determines the descriptor of a substring of a string described by D1. This substring begins in I^{th} character and its length is a function of Z. If Z is nonnegative and not greater than the string referenced by D1 then Z is itself the length of the wanted substring; otherwise if Z is negative the remainder of the string starting from the I^{th} character is assumed.

All other invalid specifications of arguments I an Z imply a null substring.

ALGORITHM SUBSTR

```
C   Let L and T be the components of D1.  TT is an auxiliary
    variable
    L = LOC(D1)
    T = LENGTH(D1)
    TT = Z
C   Check if substring is last part of D1
    IF Z < 0 THEN  TT = T-Z+1  FI
C   Validate I and Z
    IF I > 0 and 0 ≤ I + Z ≤ T + 1
       THEN SUBSTR = (L + I - 1, TT)
       ELSE SUBSTR = (L,0)
    FI
    END
```

the functions LOC and LENGTH decompose the descriptor D1, and they are described in section 5.

CONCAT (D1,D2)

This function computes the descriptor (L,T) of a string consisting of the characters of the string described by D1 followed by the characters of the string described by D2.

ALGORITHM CONCAT

```
C   Decompose D1 and D2
      L1 = LOC(D1)
      L2 = LOC(D2)
      T1 = LENGTH(D1)
      T2 = LENGTH(D2)
C   Check if one of them is the null string
      IF  T1 = 0 THEN CONCAT = D2 ; RETURN ; FI
      IF  T2 = 0 THEN CONCAT = D1 ; RETURN ; FI
      T = T1 + T2
C   If first string is on Top of the string list just
    move the 2nd string
      IF L1 + T1 = d THEN L = L1
                          i = 2
                          CALL MOVE (Di)
                     ELSE L = d
                          FOR i = 1,2
                          CALL MOVE (Di)
                          ROF
      FI
      CONCAT = (L,T)
      END
```

the routine MOVE(D_i) does the following:

```
      IF  d+Ti > M THEN CALL COMPACT FI
      FOR j = 1, Ti
          B(d+j-1) = B(Li+j-1)
      ROF
C   Change location of Di and set the new value of d
      Di = (d,Ti)
      d = d + Ti
      END
```

4 STRING COMPARISON

The functions EQ, NE, GT, LT, GE, LE compare two strings given by their descriptors and return a FORTRAN logical value equal to .TRUE. if the relation - indicated by their name is true or .FALSE. otherwise.

In the comparison the collating sequence of the characters of C is assumed. The strings are compared character by character, only the necessary to decide whether the proposed relation holds or not.

ALGORITHMS FOR COMPARISON

Let L_i = LOC(D_i) and T_i = LENGTH(D_i) for i = 1, 2

EQ(D_1,D_2)

```
EQ = .FALSE.
IF T1 ≠ T2 THEN RETURN FI
FOR j = 1, T1
    IF B(L1+j-1) ≠ B(L2+j-1) THEN RETURN FI
ROF
EQ = .TRUE.
END
```

GT(D_1,D_2)

```
C   Find the smaller in length
        i = 1
        T = T1
        IF T1 > T2 THEN T = T2
                        i = 2
        FI
C   Compare character by character until they are different
```

```
      FOR j = 1,T
         IF B(L1+j-1) ≠ B(L2+j-1)
            THEN
                IF B(L1+j-1) > B(L2+j-1) THEN GT = .TRUE.
                                         ELSE GT = .FALSE.
                FI
                RETURN
         FI
      ROF
C     If they are equal up to T decide by their length
      IF i = 2 THEN GT = .TRUE.
               ELSE GT = .FALSE.
      END
```

The other functions can be easily derived from these two presented above.

5 OTHER FUNCTIONS

Some functions which compute certain properties of strings are very useful for their manipulation.

LOC(D_1) and LENGTH(D_1)

These functions separate the components of the descriptor given as their argument.

For the sake of efficiency, these functions and some others which manipulate the components of a descriptor - are implemented in Assembler. Nevertheless they can be implemented in FORTRAN in a number of different ways as suggested in section 7.

As an example, a simple -though not efficient - manner to implement these functions is the following:

```
      INTEGER FUNCTION LOC(S)
         INTEGER S
C     THIS FUNCTION ASSUMES S TO BE A FULLWORD (32 BITS) DIVIDED
C     INTO A LEFT HALF REPRESENTING LOC AND THE OTHER REPRESENTING
C     THE LENGTH OF S
         LOC = S/32768
         RETURN
         END
      INTEGER FUNCTION LENGTH(S)
         INTEGER S
C     LENGTH IS THE REMAINDER OF THE DIVISION OF S BY 32768
         LENGTH = S-S/32768*32768
         RETURN
         END
```

Another function quite useful for the manipulation of strings

INDEX (D_1,D_2)

This function finds the position of the first occurrence (if any) of the substring given by D_2 in the string given by D_1. In case of no occurrence of the substring the value returned is zero.

ALGORITHM INDEX

```
C     Decompose D1 and D2
      FOR i = 1,2
         Li = LOC(Di)
         Ti = LENGTH(Di)
      ROF
```

```
C   Check validity of T2
        INDEX = 0
        IF T2 > T1 THEN RETURN FI
C   Check the possibilities
        FOR j = L1, L1 + T1 - T2
            S = (j, T2)
        IF  EQ(S,D2) THEN INDEX = j - L1 + 1
                          RETURN   FI

        ROF
        END
```

Similarly we can also define many other functions of strings. Those presented here are considered to be very useful in most applications.

6 STORAGE MANAGEMENT

As the strings are created the available space of B is serially occupied. Obviously this space may eventually be not enough to store a string. However not necessarily every string has ever been created is needed forever. When a string generator function computes a descriptor, this descriptor is only a value of a function-as 0.5 is the value of SQRT(0.25) for example. The value of a function is temporary and has to be recalculated every time it is referenced by the function call. The descriptors calculated by the string functions are also temporary in this sense. Only those which are stored in vector D - by an assignment statement for example are permanent. Therefore, only these strings are needed at a time, except for the temporary strings allocated by the current statement in a program.

Ex Suppose S_1 and S_2 were declared as descriptors.

S_1 = CONCAT (S2, STRING ('BLUE'))
S_2 = CONCAT (STRING ('BLA'), STRINGS ('BLUE'))

The strings 'BLUE' and 'BLA' are temporary and the second reference to 'BLUE' implies in a second allocation. Therefore the instances of string constants leave available space between the permanent strings in sector B.

It is also important to notice that strings may overlap - because of the SUBSTR function for example - so that, after some manipulation, the set of strings in B may be in the situation depicted by fig. 1.

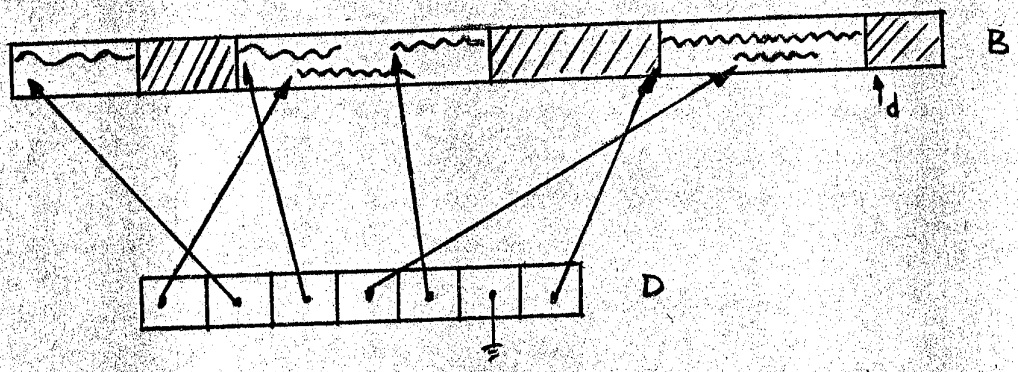

fig.1

As fig. 1 shows, the strings are partitioned in disjoints sequences of strings with no available space between them which we will call "string-compounds".

The algorithm compact presented below reorganizes the pool of strings so that, after the execution of the algorithm, the situation can be depicted as in fig. 2.

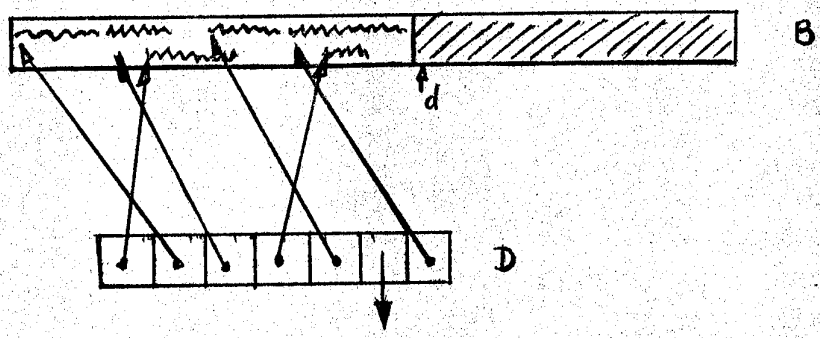

fig 2

Basically, algorithm COMPACT takes string - compounds from left to right, each string - compound being shifted to the leftmost available position in B and each group of associated descriptors in D being adjusted accordingly.

The simplicity of the technique devised for implementing the strings permits this algorithm to be rather simple and therefore it is clained be better than the one described by Hanson [2] where an additional temporary area is needed during the execution of the algorithm.

ALGORITHM COMPACT

```
C   F and U are respectively the lower and upper limits of a
C   string - compound.     is the distance to the left that
C   this string - compound has to move.
        F = 0
        U = 0
        δ = 0
        FOR j = 1,N
C   Find the least positive L_i of the descriptors. (See below.)
            i = 1
            FOR r = 1,N
                IF L_r > 0 and L_r < L_i THEN i = r FI
            ROF
```

```
C   Check if string i begins a new compound
        IF L_i > U + 1
           THEN
              Shift last compound to the left
                IF δ > 0 THEN
                              FOR r = F, U
                                 B_{r-δ} = B_r
                              ROF
                FI
C   Redefine F, U and δ
        δ = δ + L_i - (U + 1)
        F = L_i
        U = L_i + T_i - 1

        ELSE
C   This string belongs to the compound.  Just redefine U
        IF  U < L_i + T_i - 1 THEN U = L_i + T_i - 1 FI

        FI
C   Correct L_i.  Notice that δ is the same for the strings
C   of the same string-compound
        L_i = L_i - δ
C   Change sign of L_i to prevent it to be chosen again as
C   the least
        L_i = - L_i

        ROF
C   Shift the last string-compound
        IF δ > 0 THEN
                  FOR  r = F, U
                     B_{r-δ} = B_r
                  ROF

        FI
```

```
C   Reset the signs of the L_i's
        FOR j = 1,N
            L_j = -L_j
        ROF
C   Correct d if now the available space is greater
        IF U - δ + 1 < d THEN d = U - δ + 1
                         ELSE "give up!"
        FI
        END
```

The only temporary strings which may still be needed after the compaction described above are necessarily in the last shaded area of fig. 1. Because d is redefined then the creation of a new string may overlap a temporary string still needed and consequently some operations which deal with the characters and not only with the descriptors of strings may go wrong. Therefore the user must avoid more than one call to the functions STRING and GETSTR in the same statement.

7 IMPLEMENTATION

As it was pointed out before though the algorithms presented refer to two vectors D and B they will be in fact parts of one same vector. This vector will be defined in COMMON in the main program and will be accessed by very string routine.

The first three positions of this vector will be reserved for control. The following N positions (N is the number of strings in D) will be equivalent to vector D. The remainder is equivalent to vector B.

The descriptors will be INTEGER x 4 values of FORTRAN IV and their components will be extracted by functions LOC and LENGTH and set by subroutines SETLOC and SETLEN. These subroutine can also be implemented in FORTRAN though they have been made in Assembler.

For example if we use the same technique used in LOC example:

```
SUBROUTINE SETLEN(D,T)
   INTEGER D,T
   D=D/32768*32768 + T
   RETURN
   END
```

Besides this technique of building a descriptor by

```
32768*LOC + LENGTH
```

others techniques can be devised for example:

- small Assembler routines can improve the efficiency of these operations
- In some FORTRAN IV Compilers it is permitted to equivalence INTEGER*2 with INTEGER*4 variables therefore these operations can be easily done. For example the function LENGTH could be implemented as follows:

```
INTEGER FUNCTION LENGTH(D)
   INTEGER D
   INTEGER*2 C(2)
   EQUIVALENCE(D,C(1))
   LENGTH=C(2)
   RETURN
   END
```

Because we have to simulate a new data type in FORTRAN without any modification in the compiler or preprocessing we must follow certain disciplines. The area for the pool of strings and the strings themselves must be declared in the beginning of a program and certain initializations performed.

The following steps must be taken to use the facilities of character strings manipulation in a FORTRAN program:

I) Declare as INTEGER all descriptors of strings as well as all string functions used in the program.

II) Declare as LOGICAL the string comparison functions (EQ, LT, etc) used in the program.

III) Declare a COMMON area called MEMSTR for the pool of

string. The statement should be as follows:

```
COMMON /MEMSTR/M(500)
```

The name of the vector need not be M but must be an INTEGER FORTRAN variable. The dimension for this vector must be estimated by the user. It must be greater than:

> Number of permanent strings + 3 + (total number of characters needed)/4

IV) Declare that the descriptors of the permanent strings are in EQUIVALENCE with a part of area MEMSTR starting at the 4^{th} position.

For example, suppose we need four strings described by S, W, X and Y respectively. We should write:

```
----------
INTEGER S, W, X, Y
COMMON/MEMSTR/MS(200)
EQUIVALENCE (MS(4),S),(MS(5),W),(MS(6),X),(MS(7),Y)
----------
```

V) Finally in the beginning of the program (main program only) before any reference to strings write the following initialization.

```
M(1) = dimension of M
M(3) = number of permanent strings
M(2) = M(3) + 4
FOR  j = 4, M(2)
     M_j = 32768*M(1)
ROF
```

These statements set the first three positions of M to control values and initialize the permanent strings as null.

Of course the program must be link-edited with the string routines to be executable. These routine will be in a private library-named CHARLIB for example - which must be concatenated to the standard FORTRAN library.

Assuming that the standard FORTRAN library is called SYS1.FORTLIB and using the standard OS/JCL procedure FORTGCLG a simple program can be compiled, link-edited and executed with the following commands.

```
// JOB COMMAND
// EXEC FORTGCLG
// FORT.SYSIN   DD  *
          FORTRAN program which uses strings
// LKED.SYSLIB  DD  DSN=SYS1.FORTLIB,DISP=SHR
//         DD   DSN=CHARLIB, DISP=SHR
// GO.SYSIN    DD  *
          data
//
```

The next section shows a concrete program in FORTRAN which manipulates some strings.

8 EXAMPLE

Fig. 3 shows a simple program which does the following:

a) Reads a card containing a non-null string starting at column 1 and ending by a ';'.
b) Delete all occurrences of the substring 'XYZ'
c) If the final string is 'END' stops
d) If the length of the final string is greater than 3 prints the last substring of size 3 otherwise prints the final string

For clarity the program uses the DOWHILE statement of FORTS [3] as comments and intentionally abuses of string functions.

```
C    AN EXAMPLE OF MANIPULATION OF STRINGS IN FORTRAN
C    NECESSARY DECLARATIONS
      LOGICAL NE,EQ
      INTEGER SUBSTR, CONCAT, GETSTR, STRING
      COMMON/MEMSTR/M(200)/
```

```
C   PERMANENT STRINGS USED IN THE PROGRAM
        INTEGER CARTAO,LETRA
        EQUIVALENCE(M(4),CARTAO),(M(5),LETRA)
C   NECESSARY INITIALIZATIONS.  THE STRINGS MUST ALL BE NULL
        M(1)=200
        M(2)=6
        M(3)=2
        DO 1 J=4,5
1          M(J)=32768*200
C   END OF INICIALIZATION
C
C   STRING READING
        J=1
        LETRA=GETSTR(J,1)
C   -DOWHILE NE(LETRA,STRING(';'))
100 IF(EQ(LETRA,STRING(';')) GO TO 200
        CARTAO=CONCAT(CARTAO,LETRA)
        J=J+1
        LETRA=GETSTR(J,1)
C   -OD
    GO TO 100
C
C   DELETE ALL XYZ
C
200 J=INDEX(CARTAO,STRING('XYZ'))
C  -DOWHILE J.NE.0
101 IF(J.EQ.0)GO TO 201
        CARTAO=CONCAT(SUBSTR(CARTAO,1,J-1),SUBSTR(CARTAO,J+3,-1))
        J=INDEX(CARTAO,STRING('XYZ')
C   -OD
    GO TO 101
C   PRINT A SUBSTRING
C
201 IF (EQ(CARTAO,STRING('END'))STOP
    J=LENGTH(CARTAO)
    IF(J.GT.3)CARTAO=SUBSTR(CARTAO,J-2,3)
```

```
C    PRINT STRING CARTAO STARTING AT COLUMN 10
     CALL PUTSTR(CARTAO, 10)
     STOP
     END
```

ACKNDWLEDGEMENTS

I wish to thank Prof Eugenio V. Pires for his patient reading and revision of the text and Miss Elenir D. Gusmão for her kindly assistance in the implementations.

9 REFERENCES

[1] IBM Commercial Subroutine Package for IBM-1130 FORTRAN. Form N9

[2] HANSON, D.R. The manipulation of variable length string data in FORTRAN IV. Phoenix, University of Arizona, Dept of Computer Science, 1975

[3] MELO, R.N. & SCHWABE, D. Extending the control structure of FORTRAN via Macro-Generator. Rio de Janeiro,PUC, Dept. de Informática, 1976, MCSCA 1/76.