Discussion and Correspondence

Handling records with a variable structure in COBOL

J. M. Triance

Computation Department, University of Manchester Institute of Science and Technology, Sackville Street, Manchester

In many data processing applications records are encountered where the fields vary in length and position within the record. In the case of Utility Programs this variation can occur from one run to another and in the case of input data it can occur from one record to another in the same file. Normal COBOL data definition is incapable of handling this type of data but it can be handled effectively by means of the DEPENDING ON option of the OCCURS clause. But the proposed Standard in COBOL will have an adverse effect on this invaluable facility.

(Received May 1973)

1. Introduction

The DEPENDING ON option of the OCCURS clause is primarily intended for transferring variable length records to and from magnetic storage devices. For this purpose the option proves to be effective enough.

This option can, however, be just as useful for processing variable length data within the core of a computer. The straightforward use is shown in **Fig. 1** where it would be assumed that the Procedure Division coding specified would move just six 'ITEM's' to FIXED-REC. In fact, many COBOL compilers currently ignore the contents of NO-OF-ITEMS and move all 10 'ITEM's' even though, logically speaking, the last four aren't there.

```
WORKING-STORAGE SECTION.
77    NO-OF-ITEMS    PIC 99.
1     VAR-REC.
    3 ITEM           PIC X(5)  OCCURS 1 TO 10
                               DEPENDING ON
                               NO-OF-ITEMS.
1     FIXED-REC      PIC X(50).

PROCEDURE DIVISION.
      :
    MOVE 6 TO NO-OF-ITEMS.
    MOVE VAR-REC TO FIXED-REC.
```

Fig. 1

However, the proposals in CIB 16 (1973) (Tabling Handling—Substantive Change 4) require the actual record length to be used rather than the maximum length. This is both sensible and useful.

However, the full power of the DEPENDING ON option is used for handling records whose fields continually vary in length or position within the record. This situation arises frequently in data processing applications and the problem is normally overcome by restricting the facilities offered by a program or by long-winded and cumbersome coding. The use of the DEPENDING ON option in two such situations will now be demonstrated.

2. Inserting a page number in a heading (First situation)

In a utility program which lists files one of the parameters is the position in the heading where a page number is to appear. (The Utility controls the end-of-page condition and page numbering.) In **Fig. 2** it is assumed that PAGE-NO-POSITION contains this parameter and PAGE-COUNT contains the current page-number. Then the specified Procedure Division coding will insert the page number in the required position. For example, if the required position is 110, then the page number will appear in columns 110 to 112 inclusive of the HEADING record.

```
WORKING-STORAGE SECTION.
77      PAGE-NO-POSITION    PIC 999.
77      PAGE-COUNT          PIC 999.
1       HEADING.
    3     FIRST-PART.
      5   LEADING-CHARS     PIC X     OCCURS
                                      0 to 117
                                      DEPENDING
                                      ON PAGE-NO-
                                      POSITION.
    3     PAGE-NO           PIC ZZ9.
PROCEDURE DIVISION.
      :
    SUBTRACT 1 FROM PAGE-NO-POSITION.
      :
    MOVE PAGE-COUNT TO PAGE-NO.
```

Fig. 2

3. Handling variable input data (Second situation)

In a particular COBOL program the input records each consist of the name of a company followed by the names of all its subsidiaries. Because the names vary in length from three to ninety-nine characters each name is preceded by two characters which indicate the length of the field. A typical record is shown in **Fig. 3.** This consists of a firm with two subsidiaries.

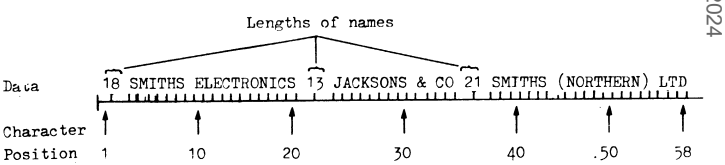

Fig. 3

This type of record is defined as INPUT-FIRMS in **Fig. 4.** Initially FIRST-PART-LENGTH contains zero which means that FIRST-PART has length zero. This NAME-LENGTH refers to the first two characters of the record which, in our example (Fig. 3), contain 18. The length of the field NAME-OF-FIRM depends on NAME-LENGTH and so is 18

characters. Thus NAME-OF-FIRM refers to characters 3 to 20 inclusive of the record and so it can be used to access the name of the firm. To access the next name the following coding must be executed:

```
ADD    NAME-LENGTH TO FIRST-PART-LENGTH.
ADD    2  TO FIRST-PART-LENGTH.
```

FIRST-PART will then be twenty characters long so that NAME-LENGTH now refers to characters 21 and 22 of the record (contents 13) and by referring to NAME-OF-FIRM in the Procedure Division the second name (JACKSONS & CO) can be accessed.

Thus by ensuring that FIRST-PART-LENGTH contains the number of characters in the record which have already been processed, each name in the record can be accessed in turn.

```
WORKING-STORAGE SECTION.
77      FIRST-PART-LENGTH      PIC 999   VALUE 0.
1       INPUT-FIRMS.
  3     FIRST-PART.
    5   FIRST-PART-CHAR        PIC X     OCCURS
                                         0 to 500
                                         DEPENDING
                                         ON FIRST-
                                         PART-
                                         LENGTH.
  3     NAME-LENGTH            PIC 99.
  3     NAME-OF-FIRM.
    5   CHAR-OF-FIRM           PIC X     OCCURS
                                         3 to 99
                                         DEPENDING
                                         ON NAME-
                                         LENGTH.
```

Fig. 4

This approach is equally applicable to any data where:

(i) the fields vary in length, or
(ii) there are many different types of field not all of which are present in any particular record.

In the latter case a code number can be used to identify the different types of field and a table in the program could indicate the length of each of them.

4. Potential drawbacks

1. *Documentation.* Although the above coding is valid ANS COBOL (1968) it is slightly artificial and therefore is not as self-documenting as COBOL is intended to be. This problem can, however, be overcome by the careful choice of data-names and the use of comments.

2. *Availability.* The techniques described above can all be implemented on the IBM 360 ANS COBOL compiler but some other compilers do not handle the coding as required. The new COBOL Standard should however clarify this situation.

3. *Core Usage.* The approach used in Fig. 4 can be wasteful on core space since the compiler must assume that all the variable parts of the record are liable to assume their maximum size simultaneously. Thus 601 characters would be reserved for the 'INPUT-FIRMS' record even though the record might never be more than 505 characters long.

To avoid this wastage a method of specifying the maximum record size would be required—a facility not currently available in COBOL. One possible solution would be to allow the PICTURE clause to be used at a group level for this purpose and in Fig. 4 write

```
01   INPUT-FIRMS   PIC X(501).
```

However, in many cases the core wastage is of little consequence.

5. Implications of proposed COBOL standard

The facility used in Figs. 2 and 4 will no longer be permitted in the proposed standard CIB 16 (1973) (Table Handling—Substantive Change 3). This states that any item which varies in length must be at the end of the record which is not the case with 'FIRST-PART' in Figs. 2 and 4.

Admittedly this restriction is proposed by CODASYL's Programming Language Committee (1972). But they provide an alternative method of handling variable length fields, namely the PICTURE character 'L', which is not included in the proposed standard.

Thus it seems reasonable that the facility demonstrated above should only be withdrawn from Standard COBOL when the alternative proposed by CODASYL has been accepted. If, on the other hand, the proposed Standard is accepted as it stands, ANS COBOL programmers will be faced with twin problems of more coding and less powerful programs.

References

COBOL Information Bulletin Number 16 (CIB 16). (1973). *The Computer Journal*, Vol. 16, No. 1, p. 81.
AMERICAN NATIONAL STANDARD COBOL—FIPS Pub. 21. (1968). American National Standards Institute.
CODASYL COBOL Journal of Development—Canadian Government Specifications Board (1970).

To the Editor
The Computer Journal

Sir

Cyclic redundancy checking by program

Higginson and Kirstein (1973) give new methods for computing a cyclic redundancy check (CRC) which give a considerable gain in speed over other methods when applied to the polynomial $x^{16} + x^{15} + x^2 + 1$ used by IBM data communication systems. It is therefore of interest to attempt to apply their methods to the polynomial $x^{16} + x^{12} + x^5 + 1$ which is recommended by CCITT (1968) and which will be used by the Post Office in its Experimental Packet Switched Service (1972).

Although the two polynomials are deceptively similar in appearance the factored form of each reveals a considerable difference in complexity namely:

IBM: $(x + 1)(x^{15} + x + 1)$
CCITT: $(x + 1)(x^{15} + x^{14} + x^{13} + x^{12} + x^4 + x^3 + x^2 + x + 1)$

The $x + 1$ term corresponds to an overall parity check which ensures that the total parity over message and CRC is even. Because of the complexity of the second term in the CCITT case it is not surprising to find that it is more difficult to apply the methods of Higginson and Kirstein to it.

Higginson and Kirstein obtain their speed gain by operating on 16 bits of the message in one process, the 'byte-pair' method. Applied to the CCITT polynomial this gives for R_{i+} (compare equation 11 of Higginson and Kirstein).

$$R_{i+1} = (M_i + R_i)(1 + x^2 + x^{12}) + (M_i + R_i)_{0-7}(x^4 + x^{12}) + (M_i + R_i)_{0-11}x^8 + (M_i + R_i)_{0-11}x + (M_i + R_i)_{0-3}(1 + x^9) + (M_i + R_i)_{0-4}x^{10} \quad (1)$$

Here the notation $(M_i + R_i)_{a-b}$ signifies