Teaching Tip

Simplifying Batch Outputting in COBOL

Barbara Russell
Computer Information Systems
Northwestern State University
Natchitoches, Louisiana 71497, USA
brussell@nsula.edu

ABSTRACT

The introduction of reference modification in standard COBOL allows COBOL programming students to bypass much of the tedium that plagues the process of report planning and programming. It is possible to avoid the drudgery of coding each heading line, detail line, subtotal line and total line in WORKING-STORAGE SECTION. Although the student will program more in the PROCEDURE DIVISION, the end result is a source program that is shorter and an object program that is smaller.

Keywords: COBOL, Report generation.

1. INTRODUCTION

The traditional method of creating output lines in COBOL is tedious and time-consuming. WORKING-STORAGE grows tremendously with each additional output line that is generated. Later, when testing, the lining up of columns of information under the heading and column heading lines can cause the student programmer many "runs" just to get the already correct output to look "pretty."

In the more recent versions of COBOL, a programming capability called reference modification ("ref mod") exists. It allows the student programmer to access specific bytes within an alphanumeric field. For example, if IN-NAME is a 6-byte field and contains JOSEPH, a programmer could code IN-NAME (1:1) and reference only the J. The parentheses contain 2 numbers separated by a colon -- the first number indicates the initial byte of the field to access and the second number shows how many bytes to access. It works similarly to the MID$ function in Visual BASIC.

By employing reference modification, the COBOL student programmer can utilize the print record defined in the FILE SECTION directly and not have to define each heading, column heading, detail line and total line called for in the report format. This greatly reduces the size of WORKING-STORAGE.

2. DISADVANTAGES

The technique does not come without some disadvantages. More code is required in the PROCEDURE DIVISION. Since the student programmer no longer pre-defines each needed output line, it must be done via MOVEs at the appropriate point in the code. But statistical analysis shows that the resulting programs are smaller than programs employing the traditional output method.

Another disadvantage associated with the reference modification method involves the problem with editing numeric values. Since reference modification assumes alphanumeric data, it will not automatically edit numeric values for output. To fix the problem, the student programmer must establish separate editing fields to first edit the value, then MOVE the edited field to the output file's print record.

3. "REF MOD" METHOD

A simple example appears in Figures 1 and 2. Figure 1 is a printer spacing chart to define the output. Figure 2 is a program that illustrates the reference modification method.

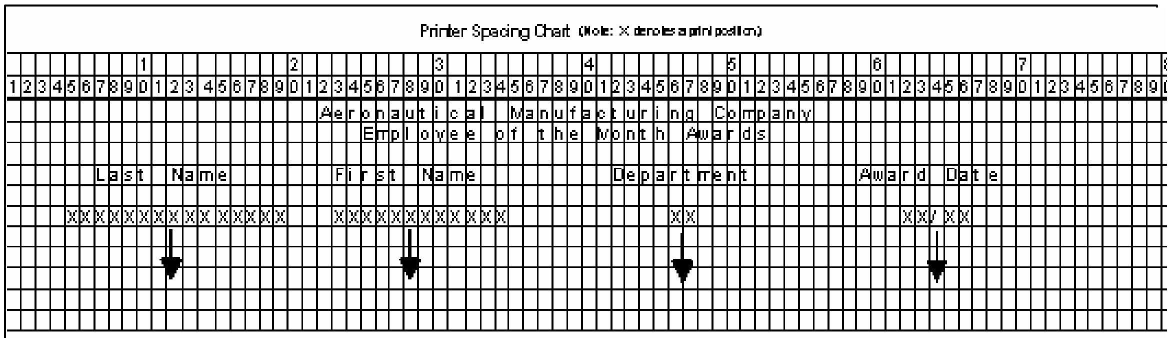

Printer Spacing Chart (Note: X denotes a print position)
Aeronautical Manufacturing Company
Employee of the Month Awards
Last Name
First Name
Department
Award Date
XXXXXXXXXXXXXXX
XXXXXXXXXXXX
XX
XX/XX

Figure 1

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  ILLUSTRATE-REF-MOD-OUTPUT.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  PC.
OBJECT-COMPUTER.  PC.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT IN-FILE ASSIGN TO DISK “C:INFILE.DAT”
           ORGANIZATION IS LINE SEQUENTIAL.
  SELECT OUT-FILE ASSIGN TO PRINTER “C:OUTFILE.RPT”.
DATA DIVISION.
FILE SECTION.
FD IN-FILE.
01 IN-REC.
  05  IN-LAST-NAME              PIC X(15).
  05  IN-FIRST-NAME             PIC X(12).
  05  IN-MONTH                  PIC 99.
  05  IN-YEAR                   PIC 99.
  05  IN-DEPT                   PIC 99.
FD OUT-FILE.
01 OUT-REC                      PIC X(80).
WORKING-STORAGE SECTION.
01  END-FLAG                    PIC XXX VALUE “NO”.
  88  DONE                       VALUE “YES”.
01 EDIT-FIELDS.
  05  ED-DATE                   PIC Z9/.
  05  ED-2-0                    PIC Z9.

PROCEDURE DIVISION.
000-START-UP.
  PERFORM 100-START-UP.
  PERFORM 200-PROCESS UNTIL DONE.
  PERFORM 300-WRAP-UP.
  STOP RUN.

100-START-UP.
  OPEN INPUT IN-FILE
       OUTPUT OUT-FILE.
  MOVE SPACES TO OUT-REC.
  PERFORM 110-HEADINGS.
  PERFORM 120-READ-FILE.

110-HEADINGS.
  MOVE “Aeronautical Manufacturing Company” TO
OUT-REC (22:34).
  WRITE OUT-REC AFTER PAGE.
  MOVE SPACES TO OUT-REC.

  MOVE “Employee of the Month Awards” TO OUT-REC
(25:28).
  WRITE OUT-REC AFTER 1.
  MOVE SPACES TO OUT-REC.

  MOVE “Last Name” TO OUT-REC (7:9).
  MOVE “First Name” TO OUT-REC (23:10).
  MOVE “Department” TO OUT-REC (42:10).
  MOVE “Award Date” TO OUT-REC (59:10).
  WRITE OUT-REC AFTER 2.
  MOVE SPACES TO OUT-REC.

  WRITE OUT-REC AFTER 1.

120-READ-FILE.
  READ IN-FILE
         AT END SET DONE TO TRUE
  END-READ.

200-PROCESS.
  PERFORM 210-DETAIL-LINE.
  PERFORM 120-READ-FILE.

210-DETAIL-LINE.
  MOVE IN-LAST-NAME TO OUT-REC (5:15).
  MOVE IN-FIRST-NAME TO OUT-REC (23:12).
  MOVE IN-DEPT TO ED-2-0.
  MOVE ED-2-0 TO OUT-REC (46:2).
  MOVE IN-MONTH TO ED-DATE.
  MOVE ED-DATE TO OUT-REC (62:3).
  MOVE IN-YEAR TO OUT-REC (65:2).
  WRITE OUT-REC AFTER 1.
  MOVE SPACES TO OUT-REC.

300-WRAP-UP.
  CLOSE IN-FILE
        OUT-FILE
```

Figure 2

As you will notice, WORKING-STORAGE is only 8 bytes long...rather unusual for a report-generating COBOL program. However, the heading and detail line modules are longer than usual. As mentioned above, when using the reference modification output method, the programmer must create each line in its entirety at the time of print. Since all print lines use the one output area (OUT-REC), nothing is left in the area that can be used during the next print.

Notice also that the output area is initialized in START-UP. This initialization is done at this point since the use of the VALUE clause is invalid in the FILE SECTION unless used with condition names. The output area is cleared to spaces following each WRITE command. This ensures that the line is clear for the next usage.

It could be argued that the student programmer could clear the line before each preparation for print. That, however, causes a problem with group indication logic employed with control break reporting where some of the print line is created in modules other than the detail line module. By initializing the output line in START-UP, then clearing it out *after* each WRITE, group indication logic can proceed without problems.

When using the reference modification method, the printer spacing chart is indispensable. The first number in the reference modification notation is the print position as shown on the printer spacing chart. This makes it easy to code and position the output as the programmer keys-in the code.

Notice how the numeric editing is handled in the third and fourth MOVEs in the DETAIL-LINE module. The numeric field is first moved to an edit field, then the edit field is moved to the output location. It is not necessary to create an edit field for each numeric field to be outputted. For instance, assume that a column of numbers is totaled. Normally the total field is larger than the detail line field. Also, normally, the total is positioned under the column so that the decimal points line up. It's possible to set-up one edit field large enough for the total field and use it for both the detail line value and the total. With this in mind, the student programmer can create a minimum of edit fields and reuse them as needed.

I have found it helpful to name the edit fields according to the number of integer and decimal positions edited. In the example program, ED-2-0 is an edit field for a 2 byte integer field -- 2 integer positions, 0 decimal positions. As I code in the PROCEDURE DIVISION, I do not have to continually turn back to WORKING-STORAGE to check field names. This is not a requirement, only a suggestion to make the numeric editing problem easier to handle.

The problem of lining up columns of information under headings and column headings becomes easier to accomplish. If one column is 3 bytes too far to the left, increasing the first number in the reference modification notation by 3 moves it. This change will not impact any of the other fields in the line, therefore eliminating the need to alter subsequent FILLER fields as in the traditional output method.

4. STATISTICAL FINDINGS

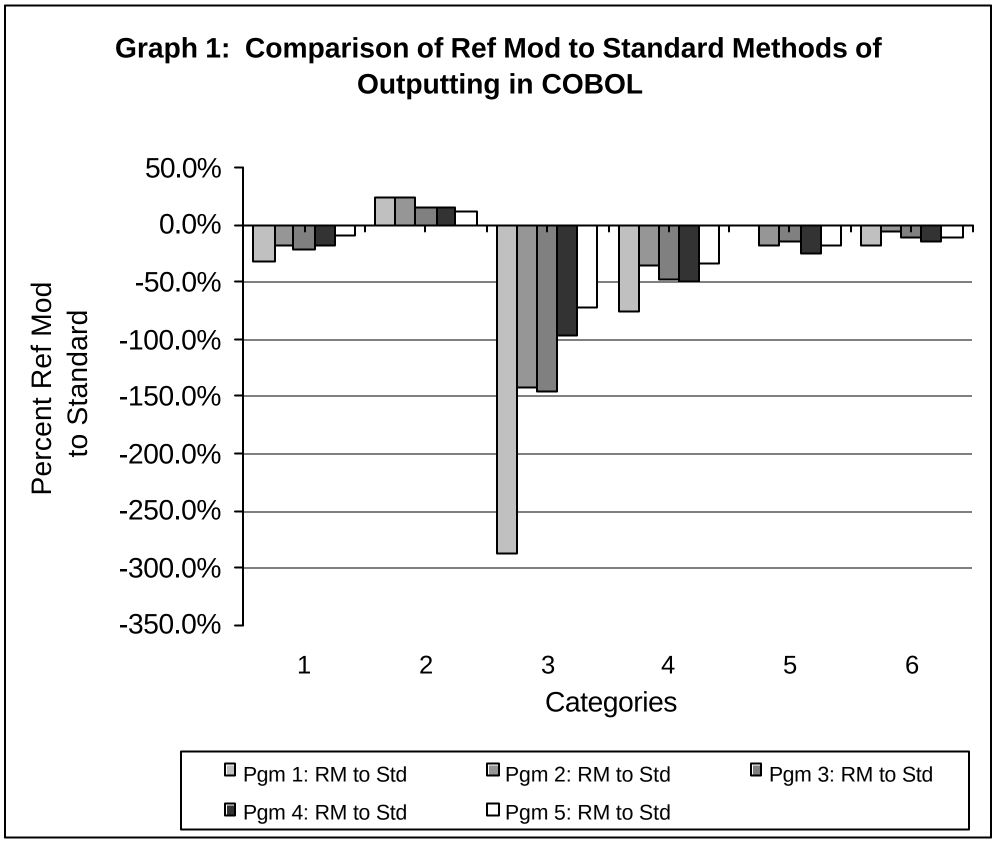
The reference modification output method was applied to five programming assignments commonly assigned in a collegiate beginning COBOL course. Characteristics of the programs follow:

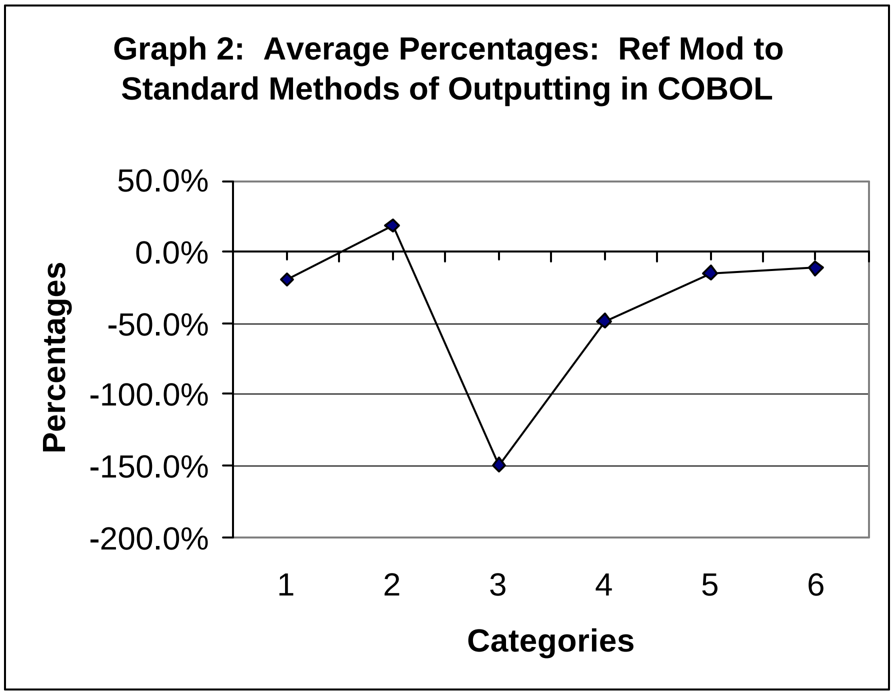
Program	Characteristics
1	One calculation for the detail line and a total
2	Four calculations for the detail line with two accumulations.
3	Single level control break, no detail line calculation, simple accumulations for subtotal and final total.
4	Two level control break, file sort, no detail line calculations, simple accumulations for subtotal and final total.
5	Detail line reporting, accumulations, and three single dimensional tables.

Each program was written using the ref mod method and the traditional method. Primary logic within the programs was maintained; only the handling of output was changed. Comparing the ref mod method to the traditional method, the following average statistics were obtained:

Category	Result
1. Total lines of code	19.1% smaller
2. PROCEDURE DIVISION	18.1% larger
3. Number of variables	149.1% fewer
4. DATA DIVISION	48.1% smaller
5. Address table space needed	14.8% smaller
6. Executable program size	11.7% smaller

Graph 1 shows the plotted percentages by program and category of the ref mod to traditional methods comparisons. Graph 2 plots the average percentages.





5. CONCLUSION

The application of the ref mod method to report generation in COBOL results in a batch programming technique that allows student programmers to get more quickly to the PROCEDURE DIVISION. The tedious WORKING-STORAGE SECTION truly becomes a portion of the program for creating working fields. This contributed to an overall improvement in attitude in the student programmers, many of whom refuse to employ the traditional method once they become comfortable with ref mod.

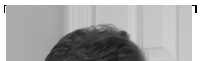
Secondly, although the PROCEDURE DIVISION is longer, comparison of programs written via the ref mod method versus the traditional method showed a decrease of 19.7% in the source program size and 11.7% smaller in the object module size. These statistics impact us in three ways: 1) smaller source and object programs require less disk storage space, 2) smaller object programs require less main memory for execution, and 3) student programmers (and their professors), on average, have 19.7% *fewer lines of code to key-in and debug*.

6. ACKNOWLEDGEMENTS

I owe special thanks to Ms. Kim Roberts of Tarleton State University for her coding expertise and to Mr. Mark Kachel who published an article, *'Cascade Method' Simplifies Programming, Improves Productivity*, Data Management, 10/86, who first caused me to think about alternative output methods.

BIOGRAPHY

Barbara Russell is an instructor of CIS at Northwestern State University in Natchitoches, LA. She worked in industry for five years before entering the teaching field and has taught for 19 years. Her areas of expertise are in programming logic, COBOL, and Visual BASIC. She holds a B.S. and M.S. in computer science and earned the CCP in 1991. Barbara coordinated six of the past seven national COBOL programming contests associated with AITP's National Collegiate Conference.



STATEMENT OF PEER REVIEW INTEGRITY

ISSN 1055-3096