The 1985 COBOL Standard

Jeff Goldberg
Dr. Domina
CSCI 465
Honors Paper
Submitted 4/19/88

The Common Business Oriented Language, COBOL, will celebrate its 30th anniversary in April of next year. The new standard for COBOL compilers, hereafter called COBOL 85, was announced in September of 1985 after seven years of deliberation. This new version of the world's most popular language presents many new improvements which will serve to further promote the efficiency of COBOL's data processing capabilities. However, nothing can be gained without sacrifice. By improving the programming efficiency this new standard had to give up complete upward compatibility.

In 1959, most programming tasks were written in machine codes. This proved to be very time consuming and costly; also few people could understand it enough to be productive. When the assemblers came out, it marked the beginning of highly productive programming. However, there still existed another major problem: an assembly language program is machine specific. Programmers had to learn almost completely different languages to work with different computers. For this purpose, and a couple of other reasons, a conference was organized. The goal of this conference, held on the campus of the University of Pennsylvania, was to recommend to the Department of Defense to have a series of meetings to come up with a, "common business language."

The following year, the Conference on Data Systems Languages, CODASYL, was held. The result was COBOL 60, the first

published standard of COBOL. The goal was to create a standard for COBOL compilers that would satisfy three main problems. The first was the problem of complicated programming. COBOL provided ease of programming, readability, and understanding to people in the computer field by using open ended English-type statements. The second problem was that of upward compatibility. In other words, the COBOL language provided for easy modifications, and programs written for an earlier compiler would be able to be compiled on the new compiler. The third and final problem to be solved was transferability. By publishing a COBOL standard, the authors wanted to be able to take a program compiled on one brand of computer and be able to compile and execute the same source code on a different computer. Thus, COBOL was officially born.

The United States of America Standards Institute, USASI, which became ANSI, merged with CODASYL to come up with COBOL 68. Since then the two groups have published COBOL 74 and COBOL 85. The International Standards Organization, ISO, has accepted ANS COBOL versions as international standards.

The rapid acceptance and continued growth in the use of COBOL had its roots in a changing programming environment. At the time of the introduction of COBOL, CPU time was relatively expensive, moreso than programmer time. As a result of this, programmers made painstaking efforts to make their programs more efficient, with little concern for the time it took to accomplish the coding task. Since that time, the roles have switched. Now, CPU time is considered cheap, and program efficiency has given

way to better documentation and ease of modification. COBOL standards have continually adjusted to this change with more descriptive reserved words and easier to incorporate modularity.

The differences between COBOL 85 and COBOL 74, the current accepted standard, are too broad for complete coverage. This new version brings with it new features, changes in old features, and deletion of obsolete features. The scope of this paper covers only the major differences that will make programming in COBOL more efficient and productive. The topics covered by this paper cut across two circles of interest; better readability of source code, and easier implementation of structured logic.

Simple things that make reading a COBOL program a little easier make up one advantage this standard has to offer. With the forty-nine new reserved words, two of them serve only to improve source code readability. When programming under COBOL 74 and describing data items as packed decimal or binary, the USAGE clause must be used with COMPUTATIONAL or COMPUTATIONAL-3. However, with COBOL 85 there are better ways to show the more efficient use of a data item. The new reserved words are PACKED-DECIMAL and BINARY. Now, data items can be declared as follows:

```
01  FIELDS.
    02  DISPLAY-ITEM      PIC S9(5) USAGE IS DISPLAY.
    02  PACKED-ITEM       PIC S9(5) USAGE IS PACKED-DECIMAL.
    02  BINARY-ITEM       PIC S9(5) USAGE IS BINARY.
```

However, even though writing out BINARY or PACKED-DECIMAL

creates better readability for debugging, the new compilers will still accept COMP and COMP-3. Eventually the latter two reserved words will be eliminated from the published COBOL standard.

Another new advantage is the number of levels a table can have. Past COBOL standards have had a limit of three levels of subscripting, while COBOL 85 has the capacity for an additional four levels. A table may be described as follows:

```
01  NEW-TABLE-TYPE.
    02  LEVEL-1                                OCCURS 5 TIMES.
        03  LEVEL-2                            OCCURS 5 TIMES.
            04  LEVEL-3                        OCCURS 5 TIMES.
                05  LEVEL-4                    OCCURS 5 TIMES.
                    06  LEVEL-5                OCCURS 5 TIMES.
                        07  LEVEL-6            OCCURS 5 TIMES.
                            08  LEVEL-7        OCCURS 5 TIMES.
                                09  DATA-ITEM  PIC 9(5).
```

When a program is designed to produce a report that will be seen by upper level management, which may be easily impressed by small details, little things such as the date become important. With COBOL 74 the compiler can return the date, and with some minor coding the date can be written in a a form management will like. Under COBOL 74, the date can be obtained by way of the MOVE statement and could be printed in the form 09/15/88. If a month table is included, the date could be put in the form of September 15, 1988. This form looks good to management, but COBOL 85 adds a new level to the aesthetics of the date, the DAY-

OF-WEEK reserved word. When DAY-OF-WEEK is used with the ACCEPT statement, the compiler returns a one digit character that stands for:

1 - Monday
2 - Tuesday
3 - Wednesday
4 - Thursday
5 - Friday
6 - Saturday
7 - Sunday

This digit could be used as a subscript to access a day table.

```
WORKING-STORAGE SECTION.

01  DAYS-TABLE

    02  DAYS-DEFINED.

        03  PIC X(9)          VALUE IS "MONDAY   ".
        03  PIC X(9)          VALUE IS "TUESDAY  ".
        03  PIC X(9)          VALUE IS "WEDNESDAY".
        03  PIC X(9)          VALUE IS "THURSDAY ".
        03  PIC X(9)          VALUE IS "FRIDAY   ".
        03  PIC X(9)          VALUE IS "SATURDAY ".
        03  PIC X(9)          VALUE IS "SUNDAY   ".

    02  DAY-NAME-TABLE REDEFINES DAYS-DEFINED.

        03  DAY-NAME          PIC X(9) OCCURS 7 TIMES.

77  DAY-CODE.
.
.
        ACCEPT DAY-CODE FROM DAY-OF-WEEK.

        DISPLAY "GOOD MORNING, TODAY IS " DAY-NAME(DAY-CODE).
```

An experienced COBOL 74 programmer would immediately notice the absence of the FILLER reserved word. Another one of COBOL 85's improvements over COBOL 74 is its more versatile use of the FILLER key word. In previous versions FILLER could not be used

at the group level. Also included in FILLER's new role is optionability. FILLER is an optional word, and if it is omitted the compiler assumes its presence. Compare the following examples.

COBOL 74 :

```
01  HEADER-LINE
    02  FILLER      PIC XXXX.
    02  FILLER      PIC XXXX.
    02  TOTALS      PIC 9999.
    02  FILLER      PIC XXXX.
```

COBOL 85 :

```
01  HEADER-LINE.
    02  FILLER.
        03          PIC XXXX.
        03          PIC XXXX.
        03  TOTALS  PIC 9999.
        03          PIC XXXX.
```

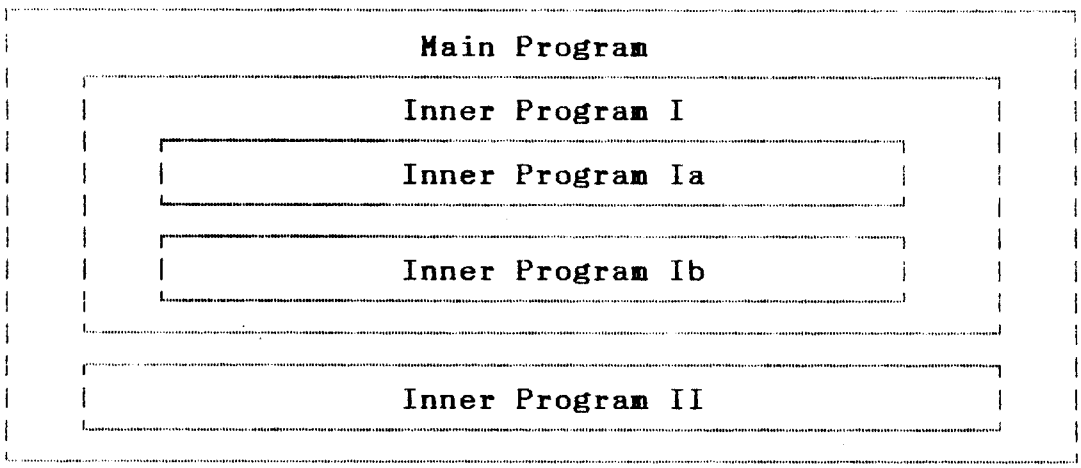
This new feature adds better readability for the programmers, and it eliminates the need to type FILLER over and over again for a program with many headers and detail lines. Since the word FILLER will be absent from the field definition, the referenceable elements will be easier to recognize.

As every programmer knows, when dealing with a program that makes use of tables of accumulators, it is necessary to use an out-of-flow loop just to initialize the fields to zero. COBOL 85 has managed to overcome this inconvenience. With this new standard, the VALUE clause can be placed on elements under an OCCURS clause. This provides a simple method to initialize a table of values to any number, as shown here:

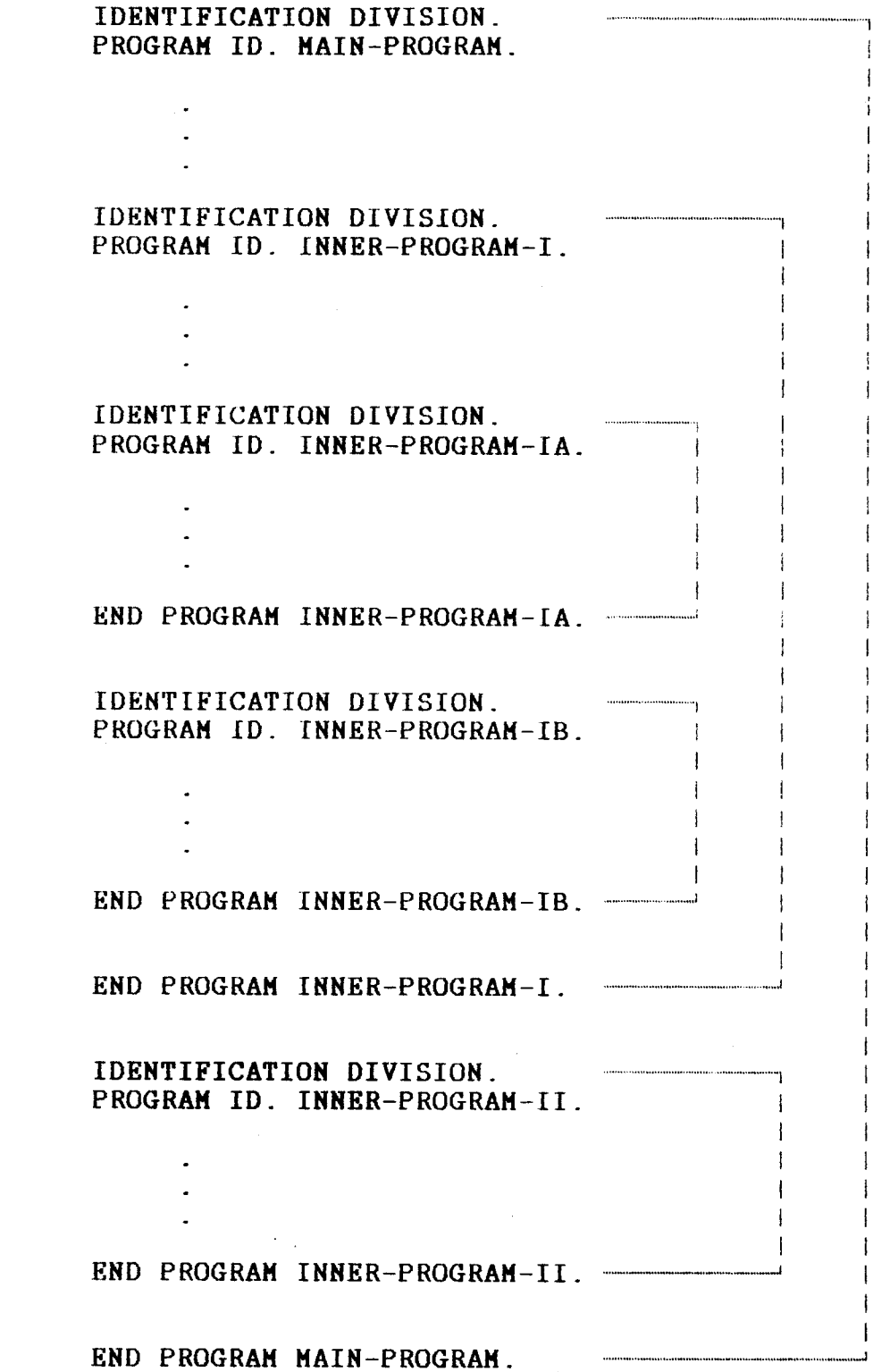
```
01.
    02  EMPLOYEE-WAGE-TOTALS              OCCURS 100 TIMES.
        03  EMPLOYEE-WAGES-TO-DATE        PIC S9(8)V99
                                          VALUE IS 0.
```

The features described so far have all resided within the circle of better readability. These small changes will make program maintenance an easier task, and in the long run, this will save valuable programming time. There is still another area to cover: better implementation of structured logic. COBOL 85 brings with it two new powerful features never before offered by COBOL. These features are nested programs and instream loops.

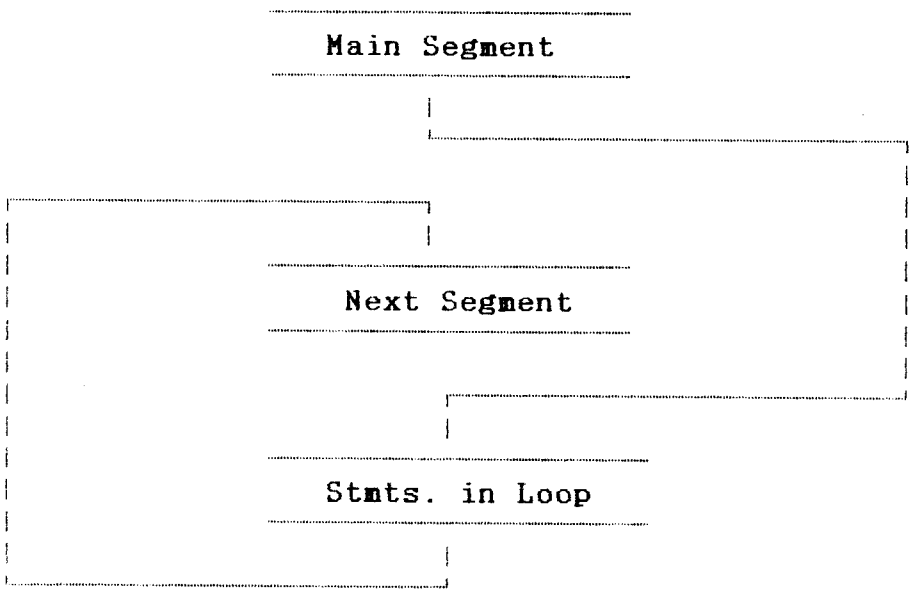
The first feature, nested programs, is very easy to implement and provides a wonderful way to use different modules created by members of a programming team. The format is conceptually, and actually, rather simple. The idea of nested programs is illustrated here:



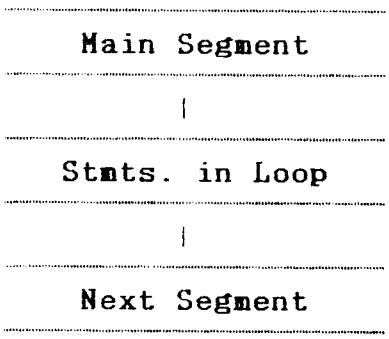
This conceptual figure can easily be transformed into a COBOL skeleton as follows:



The next new feature of COBOL 85 is probably the single greatest improvement over the previous standards. This addition is the new and improved functions of the PERFORM verb. In past COBOL standards, there was no way to process loops instream. The flow of control can be illustrated as follows:



However, with COBOL 85 the flow of control can be instream:



This difference can be seen clearly in the illustrations. All of the details of the new PERFORM statement do not need to be discussed at this time. The major aspects of the new PERFORM

verb can be seen with the following two examples.

```
PERFORM 5 TIMES
     DISPLAY ITEM(SUBSCRIPT)
     SUBSCRIPT = SUBSCRIPT + 1
END-PERFORM.
```

```
PERFORM UNTIL EOF-HAS-OCCURRED
     ADD NUMBER TO SUB-TOTAL
     READ IN-FILE AT END MOVE 'Y' TO EOF
END-PERFORM.
```

The last two features fill up the empty space in the realm of implementing structured logic. For small routines this will improve efficiency, and readability, and it will conform to structured logic. Branching to another paragraph can be eliminated, so visually the COBOL source code's flow of execution will be easier to follow, resulting in quicker program maintenance.

COBOL 85 brings with it the already mentioned features and many others, but it also has eliminated some of the old features. The list of the "obsolete" features is too long to list in its entirety, yet there is a good example of the theory behind this. In the IDENTIFICATION DIVISION there are many paragraphs that serve only as documentation. The reserved words; AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED, and SECURITY will no longer be accepted by COBOL compilers. The COBOL 85 authors'

theory behind this comes from compiler time efficiency. The compiler could save time by only having to process the necessary paragraphs. If programmers want the previously mentioned documentation in their program, they can add a comment card. Other streamlining of the compiler has taken place in the new COBOL standard.

With all of the new features, changes in old features, and the canceling of obsolete features, converting programs written for COBOL 74 to COBOL 85 is not an easy task, if it can be done at all. One of the forementioned purposes of COBOL was upward compatibility. This basic fundamental of the COBOL language has been put on the road to oblivion. The converting of present programs to COBOL 85 would take many expensive programmers long labor hours.

Most firms will not take the time to convert their old programs; thus they are faced with a tough decision. The choices for each company are (1) keep programming with a COBOL 74 compiler and ignore the new standard, (2) keep COBOL 74 and also start new programs under COBOL 85, or (3) convert COBOL 74 to COBOL 85 and code all new programs for COBOL 85. This decision is one that will be coming up very shortly in all COBOL programming environments.

The problems of converting source code to COBOL 85 from COBOL 74 appears to have the same difficulty as converting PL/I to any version of COBOL. This similarity throws a new light on the coding decision. Firms may wish to treat COBOL 85 as a

completely separate language from COBOL 74 and let each peacefully coexist in the programming environment. At the present time, this decision appears to be the most reasonable one.

Unlike previous versions of COBOL, COBOL 85 will not be rapidly implemented into mainstream programming. It may take well into the 1990's before the upgrade is finally made to be an efficient alternative to programming under COBOL 74. The slow change may cause COBOL to lose some of its popularity to newer, faster, and more powerful languages. Only time will tell.

For now, programming in COBOL will remain almost entirely under the 1974 standard. The future holds promising news for COBOL programmers in the way of new features and improvements of old features. The new standard of COBOL programming exists, yet it will be some time before the accepted standard becomes COBOL 85. Even though COBOL 85 brings with it instream logic, better defined verbs, and better documentation, COBOL 74 still stands alone at the top of the programming hill.

Works Consulted

Brown, P. R. and Gwillim V.. User's Guide to COBOL 85. Halsted Press: New York, New York. 436p. 1985.

Garfunkel, Jerome. The COBOL 85 Example Book. John Wiley & Sons: New York, New York. 322p. 1987.

I.B.M. IBM VS COBOL for OS/VS, release 2. International Business Machines: San Jose, California. 544p. 1986.

Nirmal, Barry K.. Programming Standards and Guidelines: COBOL Edition. Prentice-Hall. 226p. 1987.